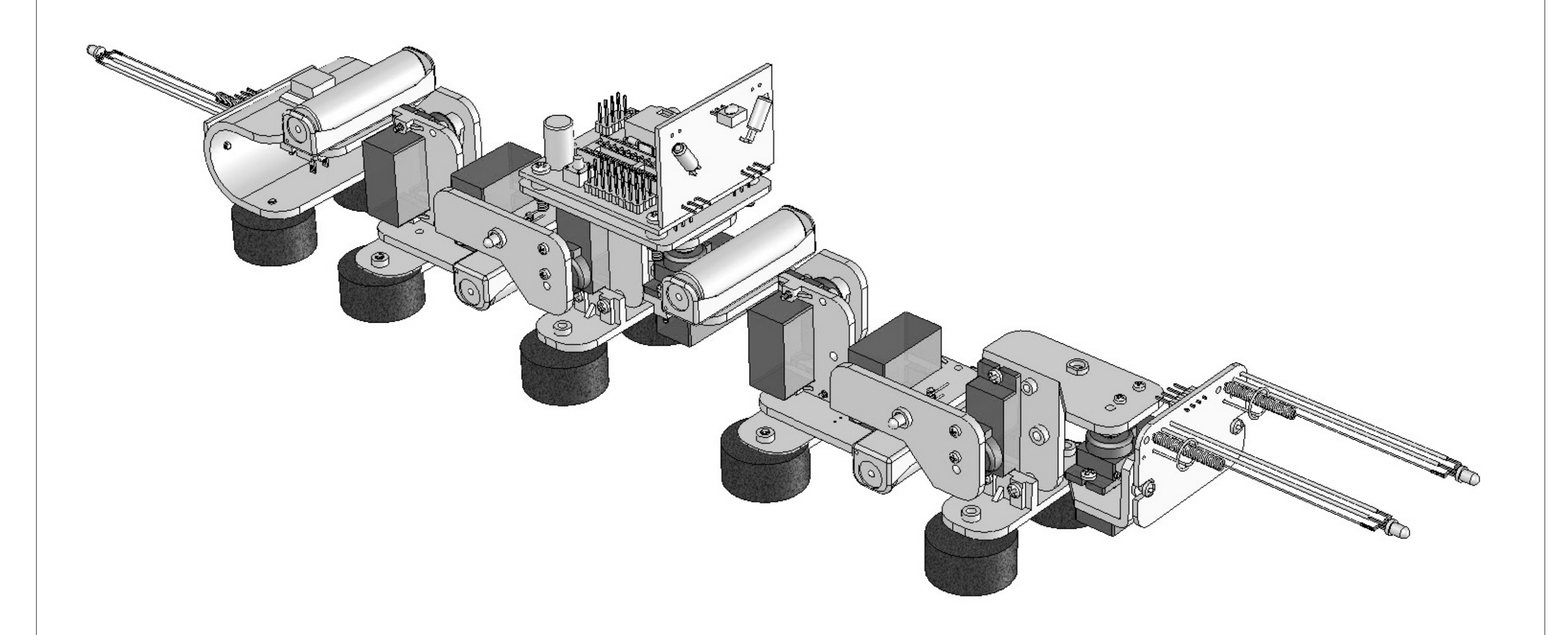
CATERPILLAR

ROBOTER-RAUPE BAUANLEITUNG



AREXX Engineering, Zwolle - NIEDERLANDE
DAGU, Zhongshan - CHINA

Inhaltsverzeichnis

1. Produktbeschreibung

Vielen Dank für die Wahl unseres Caterpillars, der acht Servomotoren, mehrere Sensoren, Elektronikteile, Hardware und Metallelemente enthält.
Es ist ein ausgezeichnetes Schulungsobjekt zum Erlernen der Grundlagen der Programmierung und Elektronik.

Produktspezifikation
1. Acht Freiheitsgrade
2. Vom Anwender programmierbar
3. I2C bus
4. Winkel-Sensor
4. Antenne-Sensoren, die Antennen wechseln die Farbe wenn diese ein Objekt berühren.
3. Zusatzanschlüsse für eine große Auswahl an AREXX Sensoren

Die Caterpillar ist sehr geeignet für Selbstlernende Programmierung un Künstliche Intelligenz Projekten.

Ehe Sie den Zusammenbau starten, empfehlen wir Ihnen die sorgfältigen Studie dieses Handbuchs. Beachten Sie bitte zur Vermeidung von Problemen genau die Bauanweisungen. Fehler beim Aufbau werden vielleicht zu Problemen im Betrieb des Roboters führen.

Spezifikation:

Betriebsspannung	: 5,2V bis 6V (4 Penlite AAA Batteriezellen zu je 1,2 oder 1,5V) (Batterien sind nicht im Bausatz eingeschlossen)
Processor	: ATMEGA16
Stromverbrauch	: ca. 1050 mA max
Höhe	: 90 mm
Länge	: 500 mm
Breite	: 60 mm

Warnung:

- Mit dem Öffnen der Plastikbeutel mit Komponenten und Teilen erlischt das Rückgaberecht.
- Lesen Sie vor dem Bauen zuerst die Gebrauchsanleitung aufmerksam durch.
- Seien Sie vorsichtig beim Hantieren mit den Werkzeugen.
- Bauen Sie nicht im Beisein kleiner Kinder. Die Kinder können sich an den Werkzeugen verletzen oder kleine Komponenten und Teile in den Mund stecken.
- Achten Sie auf die Polung der Batterien.
- Sorgen Sie dafür, daß die Batterien und die Batteriehalter trocken bleiben.
  Falls der CATERPILLAR naß wird, entfernen Sie dann die Batterien und trockne alle Teile, so gut es geht.
- Entfernen Sie die Batterien, wenn der Roboter mehr als eine Woche ruht.

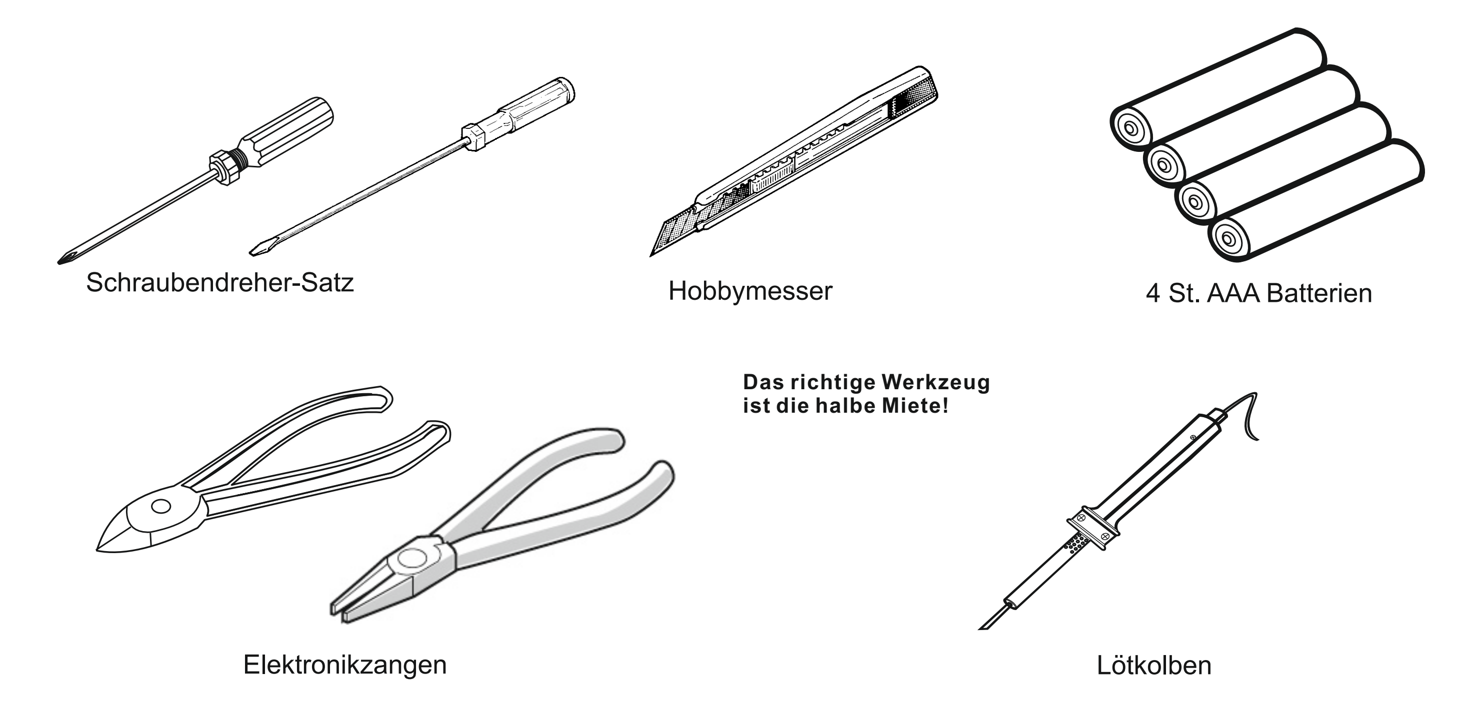
Fabrikant:
AREXX CHINA
DAGU Hi-Tech, China

Distributor:
AREXX Engineering
Zwolle, Holland

2. Werkzeug und Mechanik

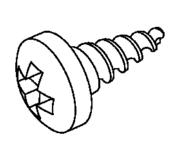
Hinweis: Lesen Sie diesen Abschnitt zuallererst durch !

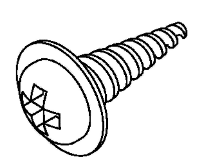
Weichen Sie bitte nicht von der Reihenfolge in dieser Beschreibung ab. Damit vermeiden Sie Montagefehler. Wer die Reihenfolge genau verfolgt und ab und zu das Foto auf der Verpackung betrachtet, baut auf Anhieb einen perfekt funktionierenden Roboter. Arbeiten Sie ruhig und lesen Sie vor Beginn der Montage diese Anleitung GANZ durch. Beachten Sie bitte zur Vermeidung von Problemen genau die Bauanweisungen. Fehler beim Aufbau werden vielleicht zu Problemen im Betrieb des Roboters führen.



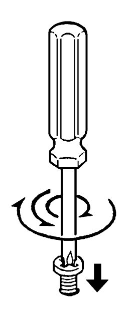
Selbstschneidende Schrauben

Schrauben mit einem selbstzapfenden Gewinde verhalten sich wie Holzschrauben, d.h. in einer Drehbewegung schneidet sich die Schraube ein Gewinde und dreht sich dabei fest in das Material. Dazu hat diese Schraubenart ein größeres Gewinde und eine schärfere Spitze als die normale Schraube.





1. Eindrehen der Schraube
2. Leichte Lockerung der Schraube
3. Anschließend wieder Festdrehen der Schraube

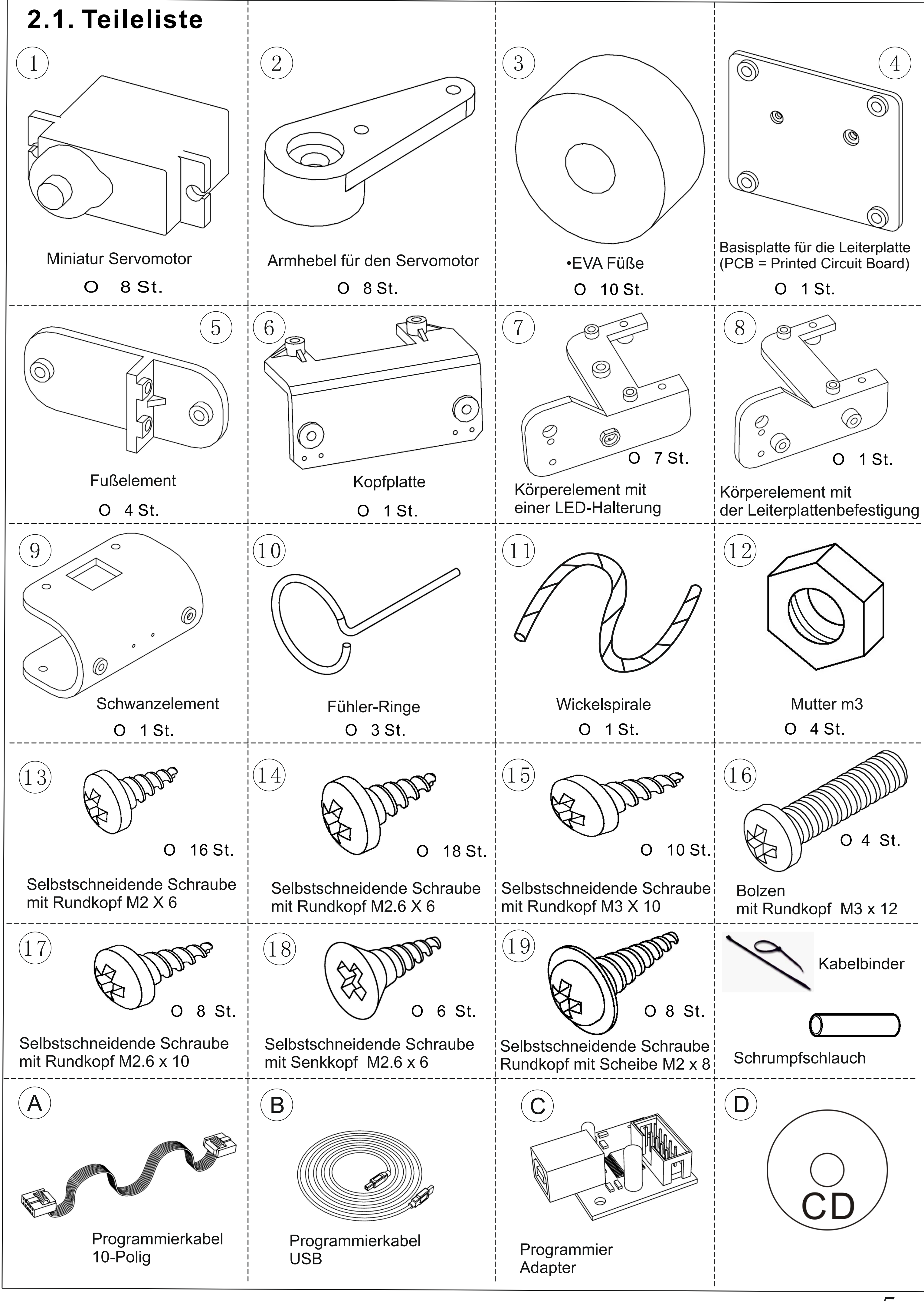


Benutzen Sie einen GUT passenden Kreuzschraubendreher, mit dem Sie kräftig die selbstzapfenden Schrauben andrehen können.

Achtung:

Falls die Schrauben zu oft gelockert und wieder festgeschraubt werden,
weitet sich das Schraubloch immer mehr und dann paßt die Schraube nicht mehr richtig.

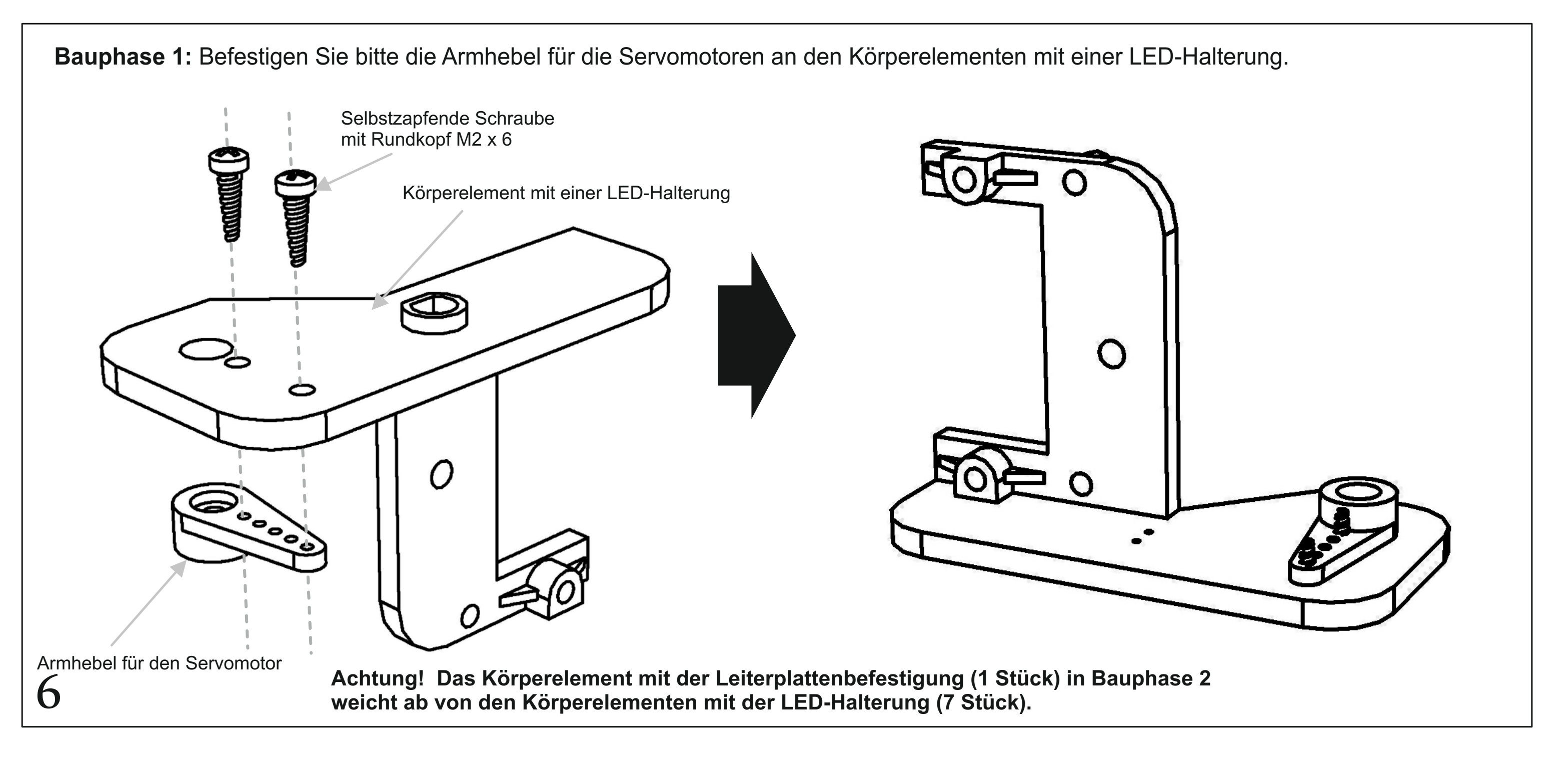
2.1. Teileliste



Nr.	Teil	Menge
20	Hauptplatine	O 1 St.
21	Leiterplatte für den Rollsensor	O 1 St.
22	Kopfplatine	O 1 St.
23	Schwanzplatine	
24	2-poliges Schalterkabel	
25	Schalter	
26	8-polige Fassung mit Flachbandkabel für die LEDs	
27	Zweifarbige LED einschließlich Feder	O 3 St.
28	Rote LED	O 4 St.
29	Anschlusskabel für den Miniatur-Servomotor	O 2 St.
30	3-poliges Anschlusskabel für Schwanzplatine	
31	4-poliger Anschlusskabel für Kopfplatine	
32	Batteriehalter	O 4 St.
33	Empfohlener Sensor: SHARP GP2D12 **NICHT MITGELIEFERT!**	

3. ANLEITUNG MECHANISCHER AUFBAU

Bauphase 1: Befestigen Sie bitte die Armhebel für die Servomotoren an den Körperelementen mit einer LED-Halterung.



Achtung! Das Körperelement mit der Leiterplattenbefestigung (1 Stück) in Bauphase 2 weicht ab von den Körperelementen mit der LED-Halterung (7 Stück).

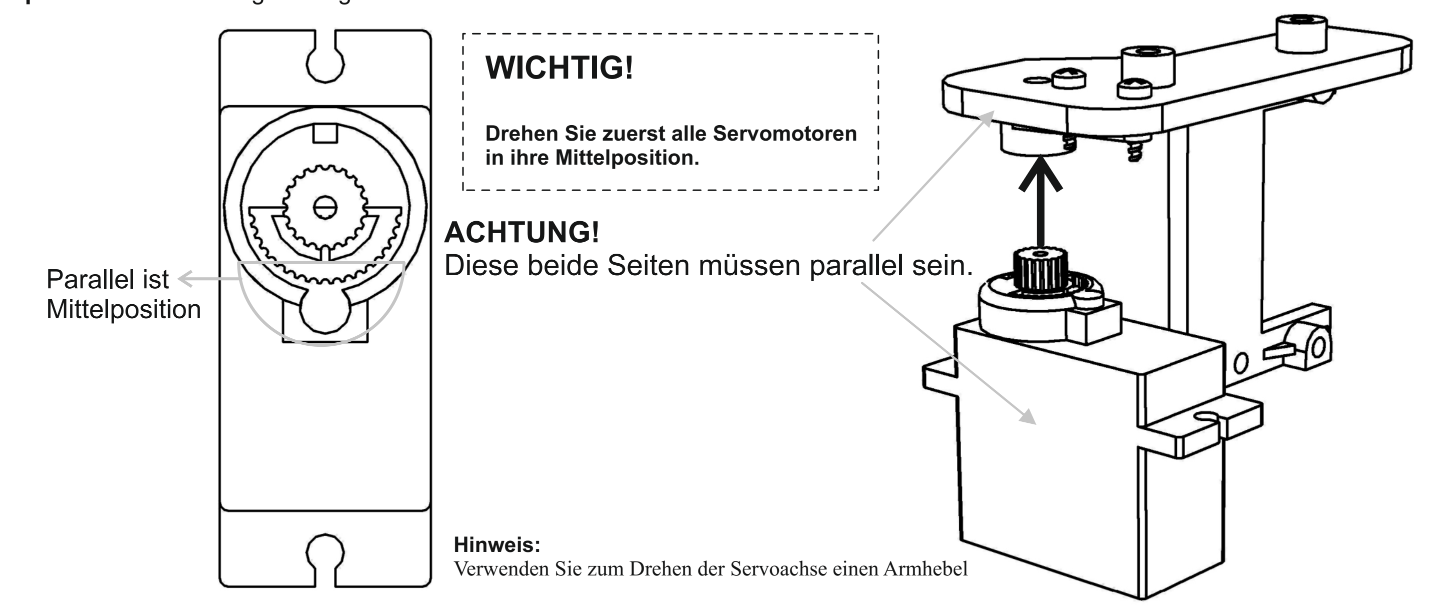
Bauphase 2: Befestigen Sie bitte den Armhebel für den Servomotor am Körperelement mit der Leiterplattenbefestigung.

Selbstzapfende Schraube mit Rundkopf M2 x 6

Körperelement *mit der Leiterplattenbefestigung*

Armhebel für den Servomotor

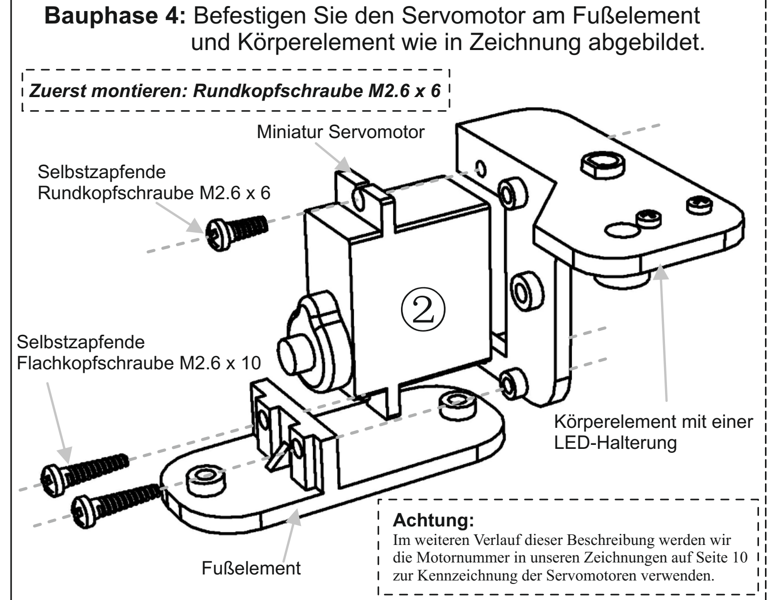
Bauphase 3: Vorbereitung Montagehinweis der Servomotoren.



Achtung: Beachten Sie bitte, dass Sie immer zuerst den Servomotor in die mittlere Zentralposition drehen müssen, ehe Sie dieses Teil am Hebelarm für den Servomotor befestigen! Befestigen Sie den Servomotor genau so wie in der Zeichnung abgebildet wird.

Bauphase 4: Befestigen Sie den Servomotor am Fußelement und Körperelement wie in Zeichnung abgebildet.

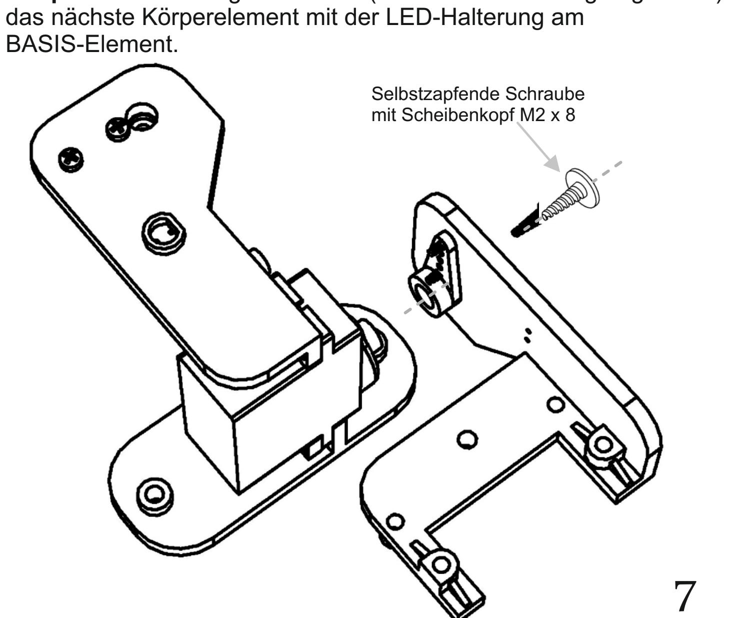
Zuerst montieren: Rundkopfschraube M2.6 x 6



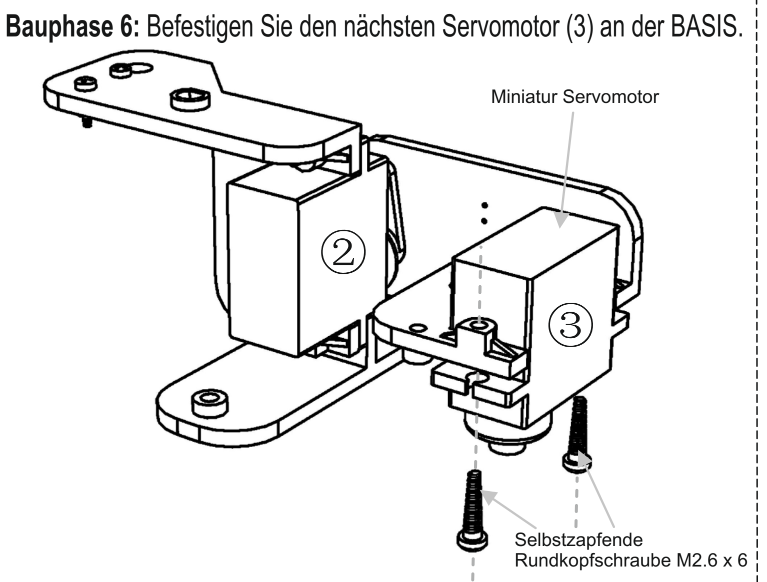
Achtung:
Im weiteren Verlauf dieser Beschreibung werden wir die Motornummer in unseren Zeichnungen auf Seite 10 zur Kennzeichnung der Servomotoren verwenden.

Ab jetzt werden wir jedes fertig zusammengestellte Element als BASIS-Element oder BASIS bezeichnen.

Bauphase 5: Befestigen Sie bitte (wie in der Zeichnung abgebildet) das nächste Körperelement mit der LED-Halterung am BASIS-Element.



Bauphase 6: Befestigen Sie den nächsten Servomotor (3) an der BASIS.



Bauphase 7: Befestigen Sie nun das nächste Körperelement mit LED-Halterung an der BASIS.

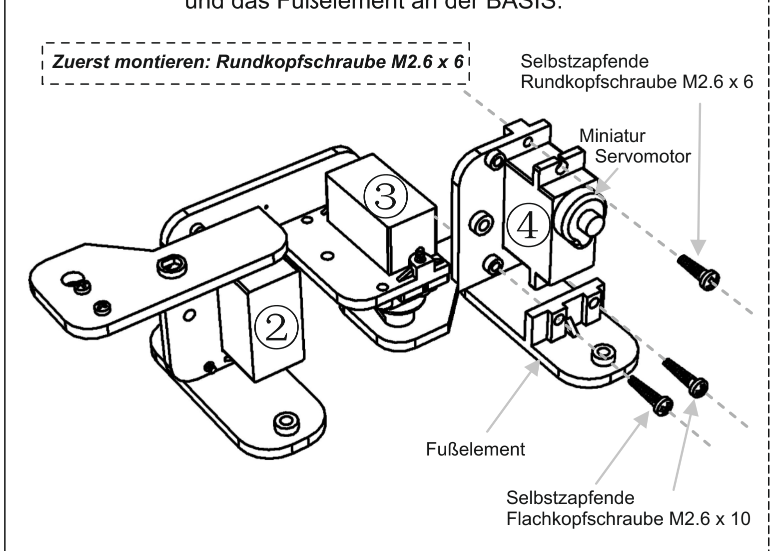
②

③

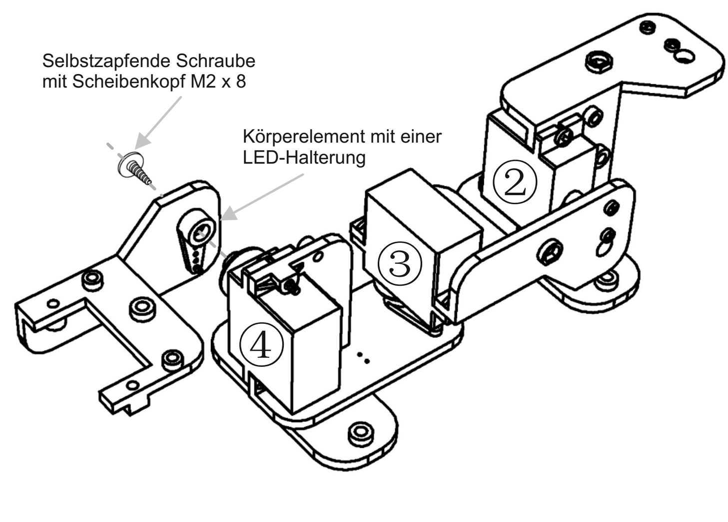
Selbstzapfende Schraube
mit Scheibenkopf M2 x 8

Körperelement mit einer
LED-Halterung

Bauphase 8: Befestigen Sie den nächsten Servomotor (4) und das Fußelement an der BASIS.



Bauphase 9: Befestigen Sie nun das nächste Körperelement mit LED-Halterung an der BASIS.



Bauphase 10: Befestigen Sie den nächsten Servomotor (5) an der BASIS.

Selbstzapfende
Rundkopfschraube M2.6 x 6

Miniatur
Servomotor

②

③

④

⑤

Achtung!
Beachten Sie bitte dass Sie dabei jetzt auch wirklich das Körperelement mit der Leiterplattenhalterung benutzen.

Selbstzapfende Schraube
mit Scheibenkopf M2 x 8

Körperelement mit einer
Leiterplattenhalterung

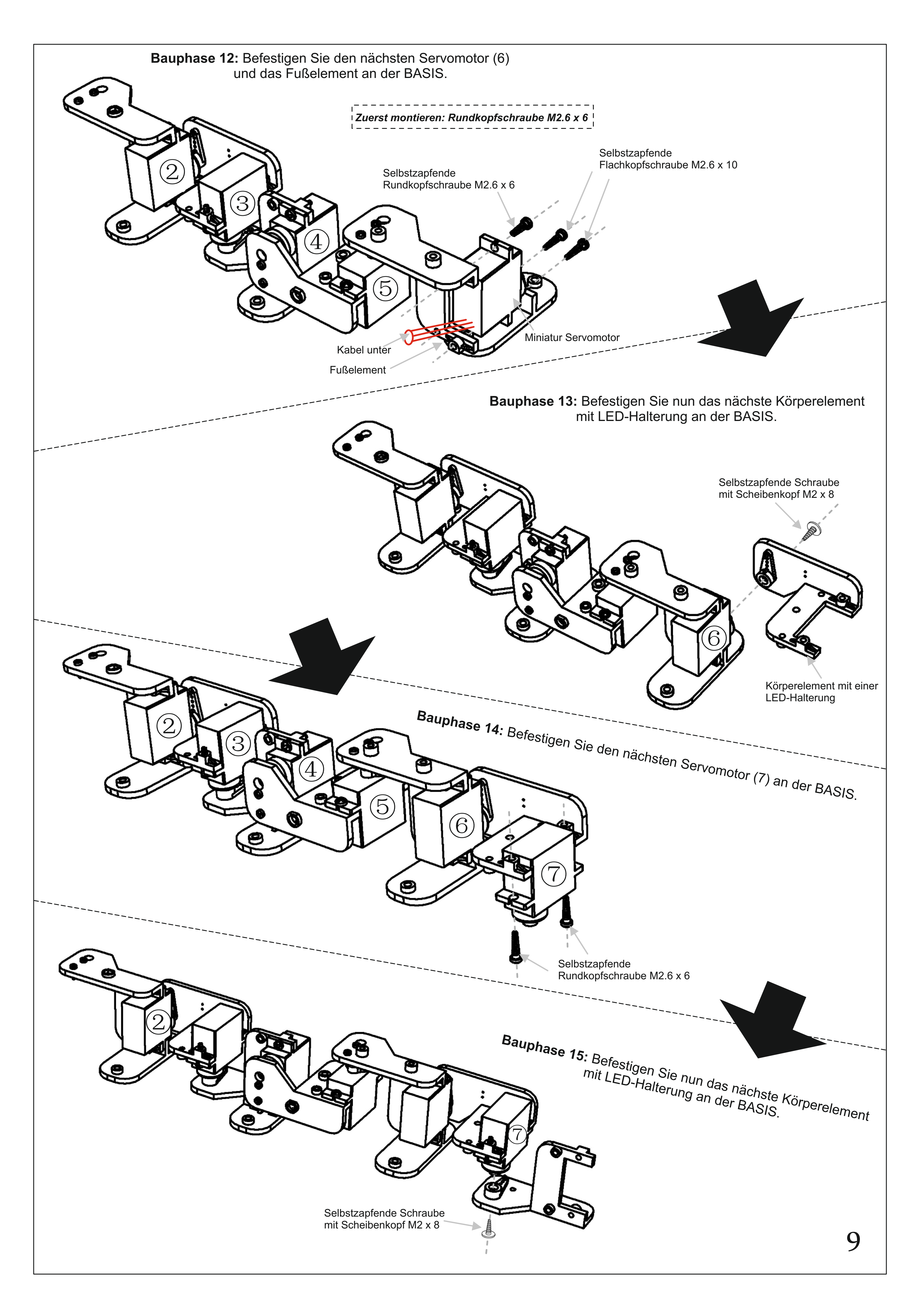
②

③

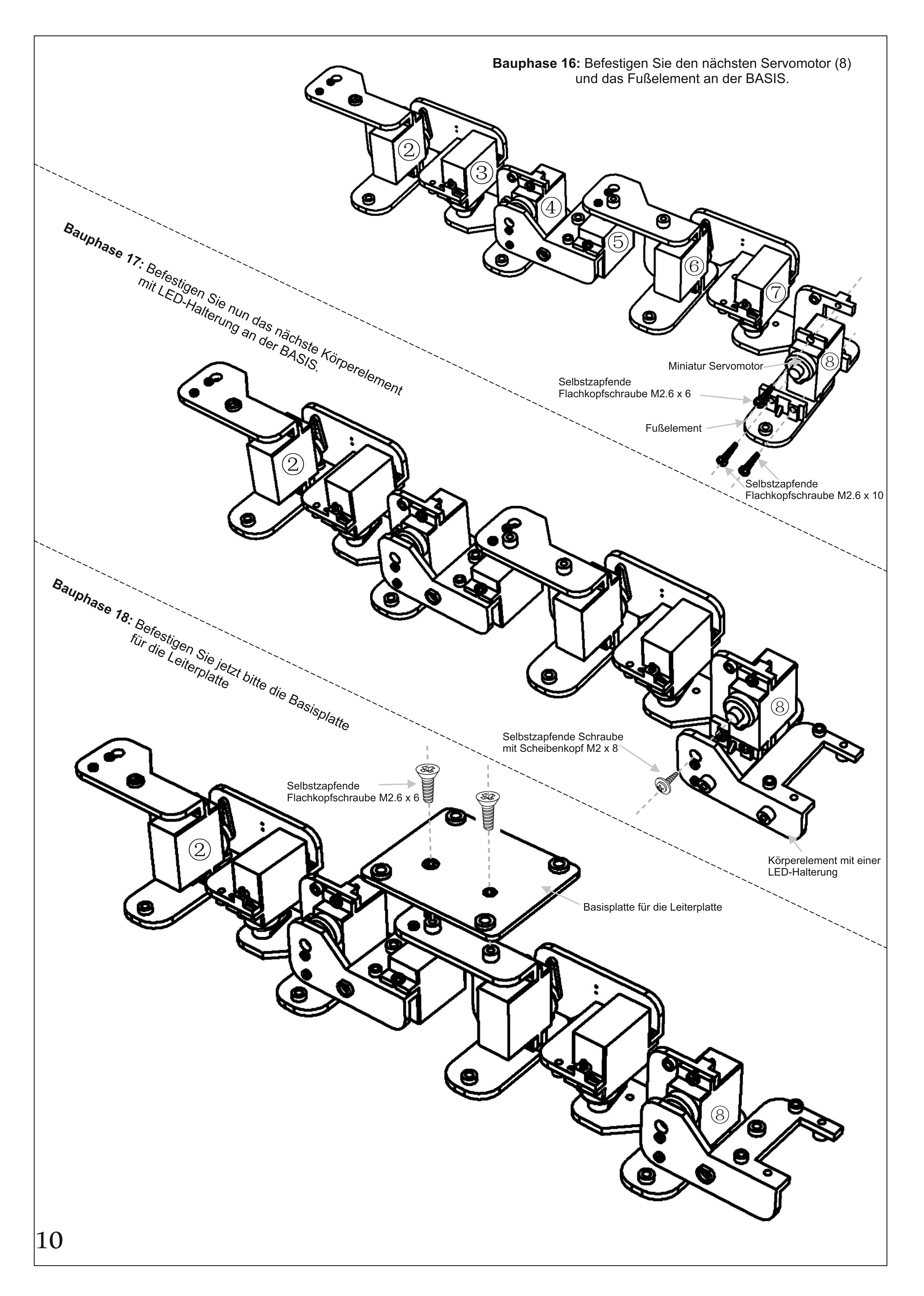
④

⑤

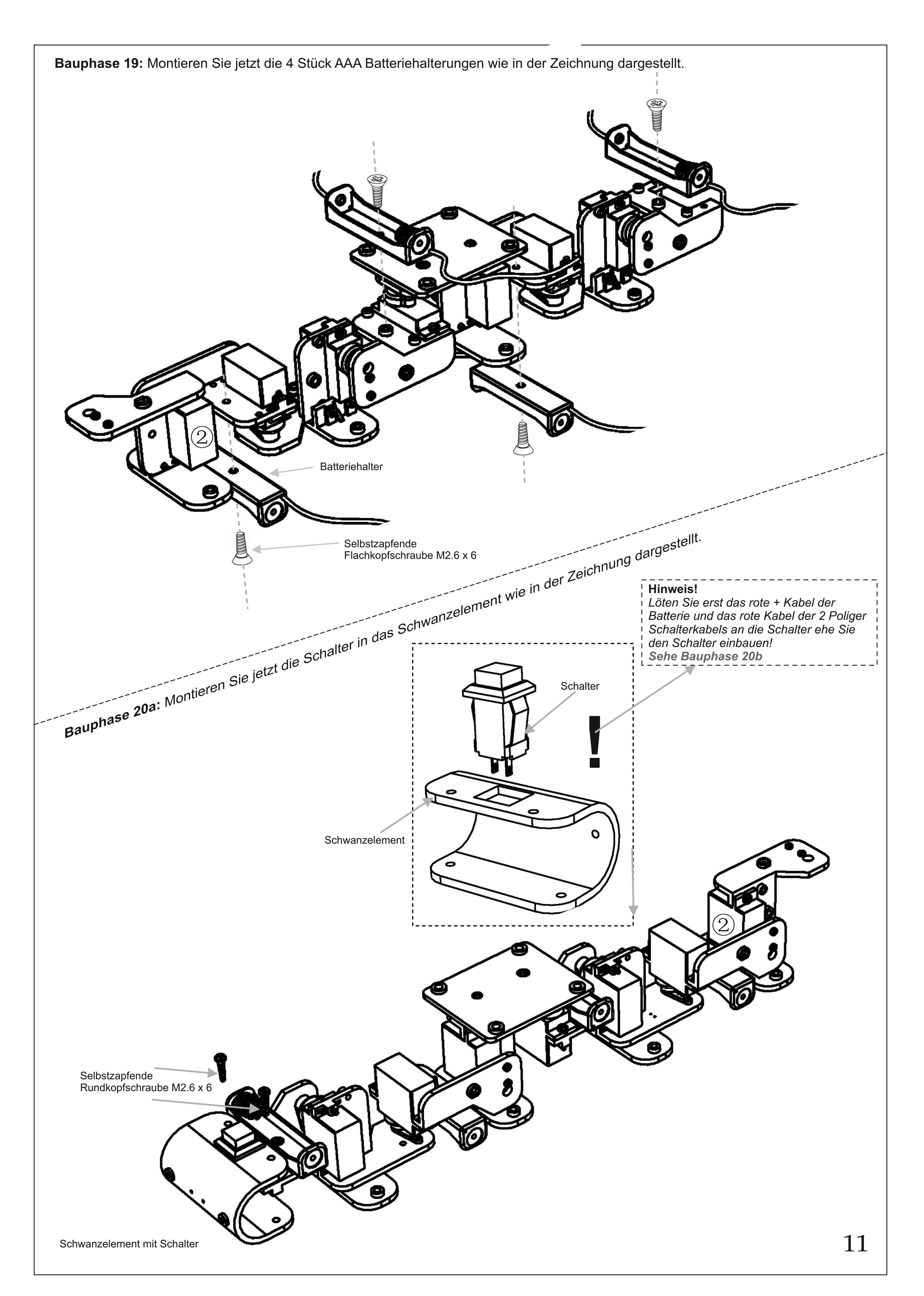
Bauphase 11: Befestigen Sie nun das nächste Körperelement mit der Leiterplattenhalterung an der BASIS.



Bauphase 12: Befestigen Sie den nächsten Servomotor (6) und das Fußelement an der BASIS.
Zuerst montieren: Rundkopfschraube M2.6 x 6
Selbstzapfende Flachkopfschraube M2.6 x 10
Selbstzapfende Rundkopfschraube M2.6 x 6
Kabel unter Fußelement
Miniatur Servomotor
Bauphase 13: Befestigen Sie nun das nächste Körperelement mit LED-Halterung an der BASIS.
Selbstzapfende Schraube mit Scheibenkopf M2 x 8
Körperelement mit einer LED-Halterung
Bauphase 14: Befestigen Sie den nächsten Servomotor (7) an der BASIS.
Selbstzapfende Rundkopfschraube M2.6 x 6
Bauphase 15: Befestigen Sie nun das nächste Körperelement mit LED-Halterung an der BASIS.
Selbstzapfende Schraube mit Scheibenkopf M2 x 8

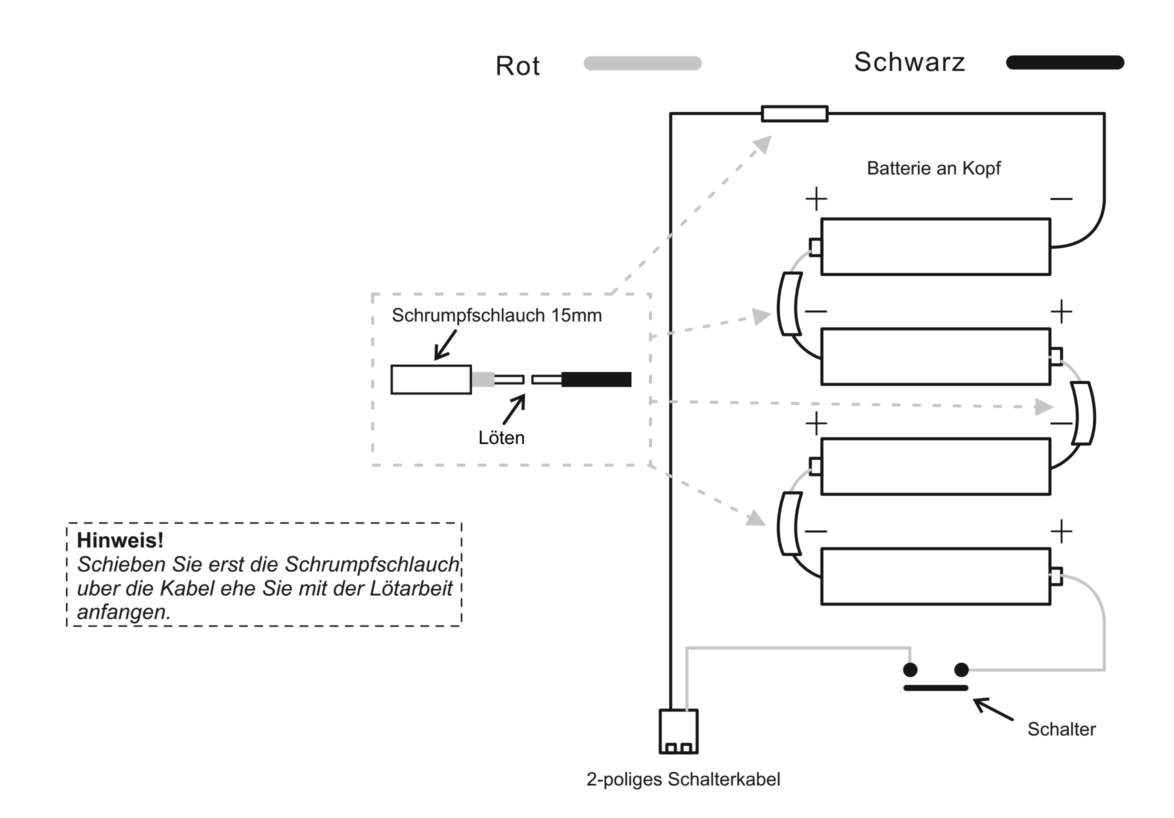

Bauphase 16: Befestigen Sie den nächsten Servomotor (8)
und das Fußelement an der BASIS.
②
③
④
⑤
⑥
⑦
⑧
Miniatur Servomotor
Selbstzapfende
Flachkopfschraube M2.6 x 6
Fußelement
Selbstzapfende
Flachkopfschraube M2.6 x 10
Bauphase 17: Befestigen Sie nun das nächste Körperelement
mit LED-Halterung an der BASIS.
②
⑧
Selbstzapfende Schraube
mit Scheibenkopf M2 x 8
Körperelement mit einer
LED-Halterung
Bauphase 18: Befestigen Sie jetzt bitte die Basisplatte
für die Leiterplatte
Selbstzapfende
Flachkopfschraube M2.6 x 6
Basisplatte für die Leiterplatte
②
⑧

Bauphase 19: Montieren Sie jetzt die 4 Stück AAA Batteriehalterungen wie in der Zeichnung dargestellt.



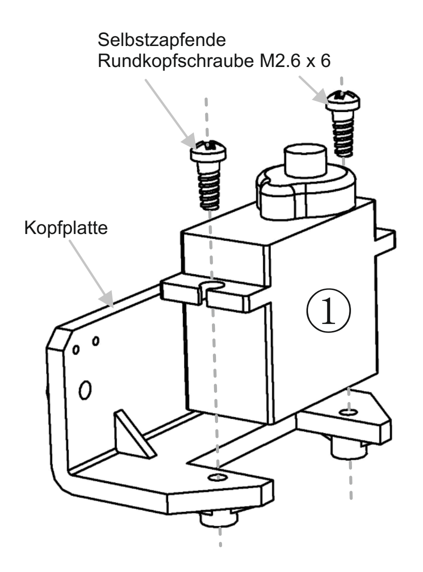
Schwanzelement mit Schalter

Bauphase 20b: Löten Sie jetzt wie in der Zeichnung abgebildet die Verdrahtungen an die Batteriehalterungen.



Achtung!
Vermeiden Sie Kurzschlüsse und Fehler in dieser Montagephase.

Bauphase 21a: Befestigen Sie den Servomotor (1) an der Kopfplatte.



Bauphase 21b: Verbinden Sie die Kopfplatte mit dem ersten Körperelement.

Selbstzapfende Schraube
mit Scheibenkopf M2 x 8

②

Kopfplatte mit
Schalter

Bauphase 22: Befestigen Sie nun die 10 (St.) EVA Füße an den Fußelementen.

Selbstzapfende
Rundkopfschraube M3 x 10

②

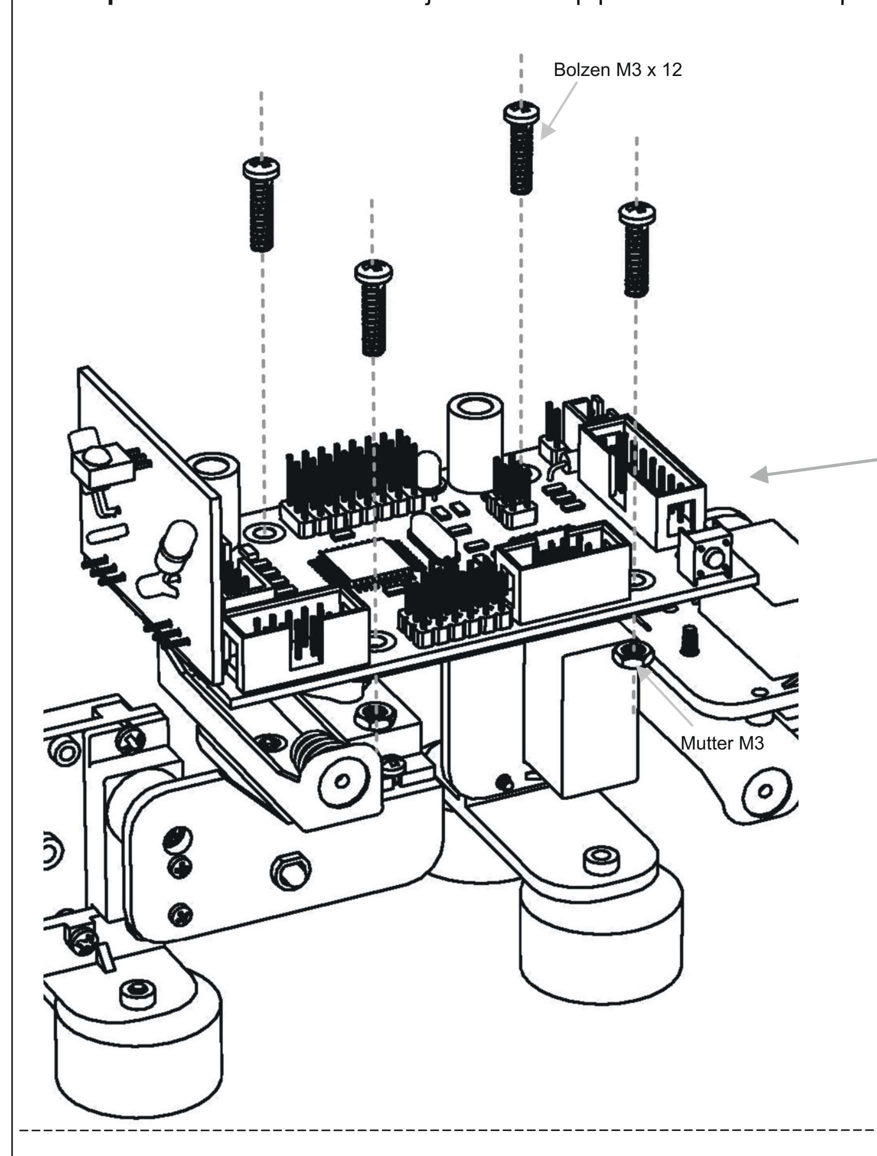
Fußplatte

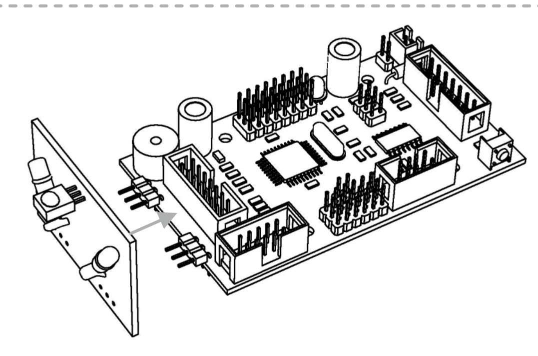
Schraube muss gleich sein
mit der Fußplatte!

EVA Fuß

Verbinden des EVA Fußes mit M3 x 10 Schrauben wie in der Zeichnung abgebildet.

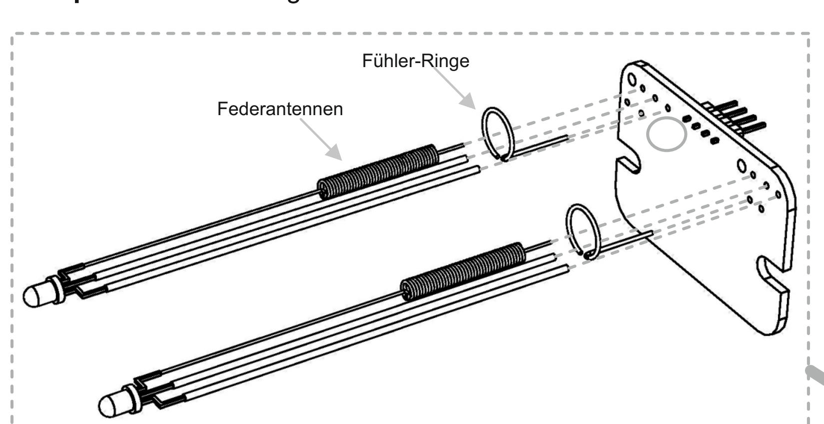
Bauphase 23a: Montieren Sie jetzt die Hauptplatine auf die Leiterplattenhalterung.





Bauphase 23b: Löten Sie die Leiterplatte für den Rollsensor auf die Hauptplatine.

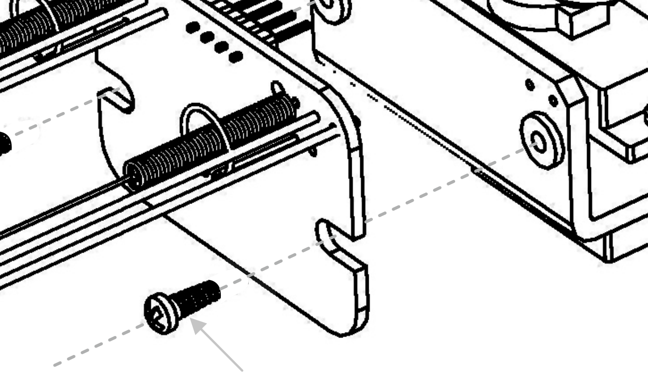
Bauphase 24: Befestigen Sie die Fühler.



Achtung!
Die Feder kann sehr scharf sein!

Löten Sie dazu die Fühler-Ringe zuerst auf die Leiterplatte und komplettieren Sie danach die Fühler wie skizziert.
Die Federantennen sollten möglichst genau in der Mitte der Fühler-Ringe montiert werden, ohne jeweils den Ring zu berühren.

Nach der Lötbefestigung der Antennen auf der Hauptplatine befestigen Sie bitte die Leiterplatte wie abgebildet auf der Kopfplatte.



Bauphase 25: Montieren Sie jetzt die Schwanzantenne.

Achtung!
Die Feder kann sehr scharf sein!

Fühler-Ringe

Federantennen

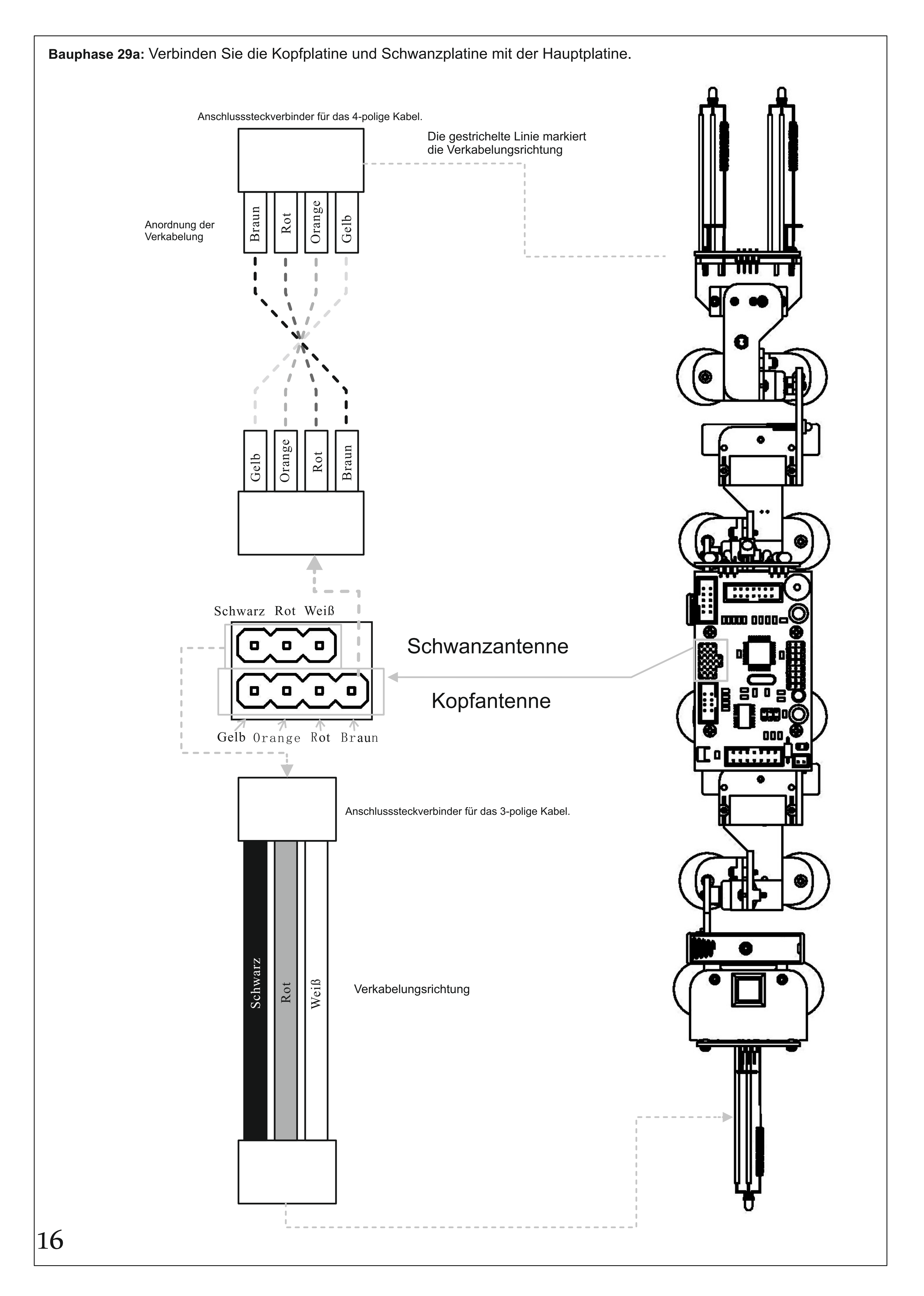
Löten Sie dazu den Fühler-Ring zuerst auf die Leiterplatte und komplettieren Sie danach den Fühler wie skizziert.
Die Federantenne sollte möglichst genau in der Mitte des Fühler-Rings montiert werden, ohne den Ring zu berühren.

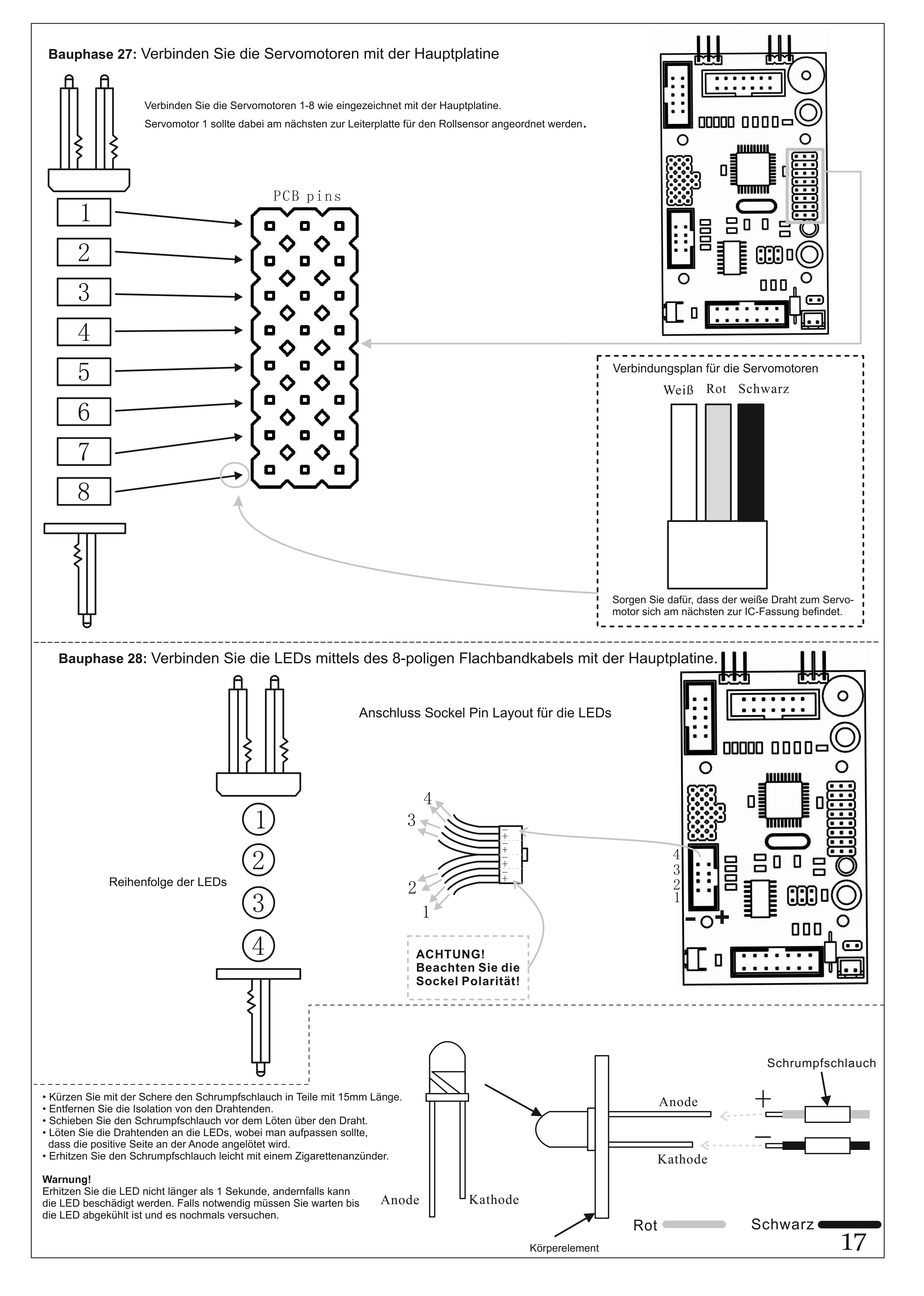
Selbstzapfende
Rundkopfschraube M2.6 x 6

Bauphase 26: Befestigen Sie die 4 (Stück) LEDs

Rote LED

Bauphase 29a: Verbinden Sie die Kopfplatine und Schwanzplatine mit der Hauptplatine.





Bauphase 27: Verbinden Sie die Servomotoren mit der Hauptplatine

Verbinden Sie die Servomotoren 1-8 wie eingezeichnet mit der Hauptplatine.

Servomotor 1 sollte dabei am nächsten zur Leiterplatte für den Rollsensor angeordnet werden.

Sorgen Sie dafür, dass der weiße Draht zum Servomotor sich am nächsten zur IC-Fassung befindet.

Bauphase 28: Verbinden Sie die LEDs mittels des 8-poligen Flachbandkabels mit der Hauptplatine.

Anschluss Sockel Pin Layout für die LEDs

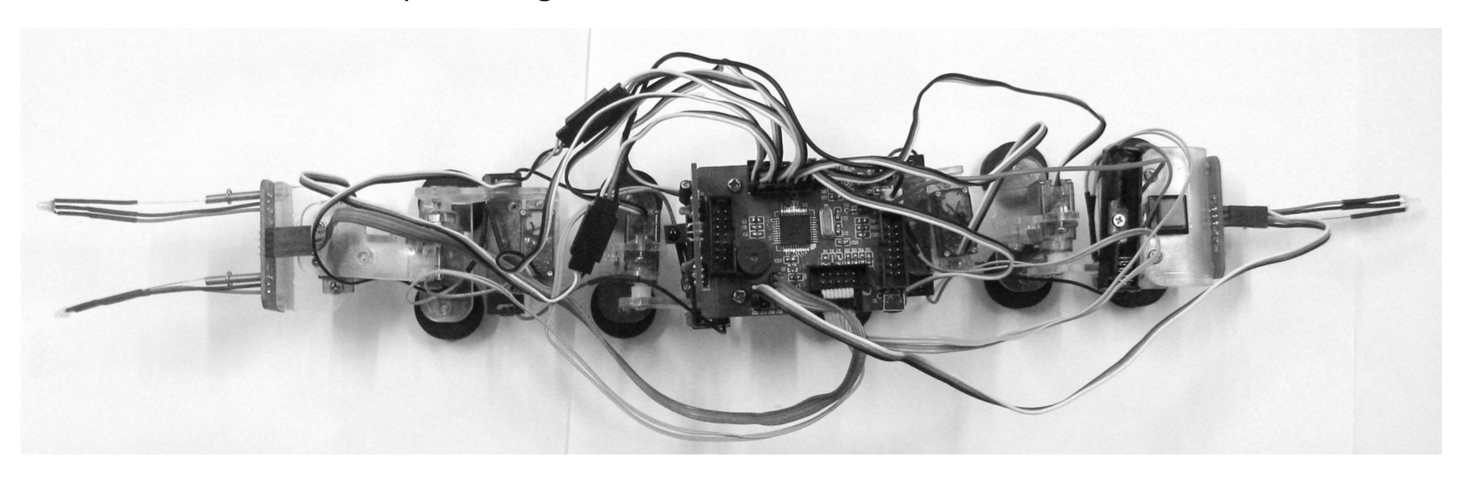
Reihenfolge der LEDs

ACHTUNG! Beachten Sie die Sockel Polarität!

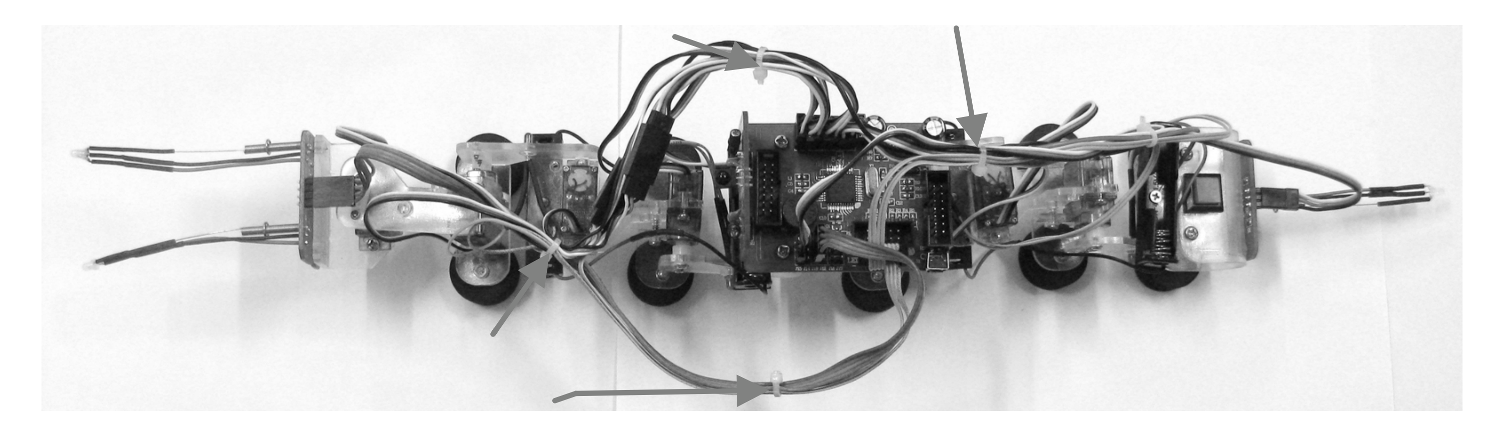
- Kürzen Sie mit der Schere den Schrumpfschlauch in Teile mit 15mm Länge.
- Entfernen Sie die Isolation von den Drahtenden.
- Schieben Sie den Schrumpfschlauch vor dem Löten über den Draht.
- Löten Sie die Drahtenden an die LEDs, wobei man aufpassen sollte, dass die positive Seite an der Anode angelötet wird.
- Erhitzen Sie den Schrumpfschlauch leicht mit einem Zigarettenanzünder.

Warnung!
Erhitzen Sie die LED nicht länger als 1 Sekunde, andernfalls kann die LED beschädigt werden. Falls notwendig müssen Sie warten bis die LED abgekühlt ist und es nochmals versuchen.

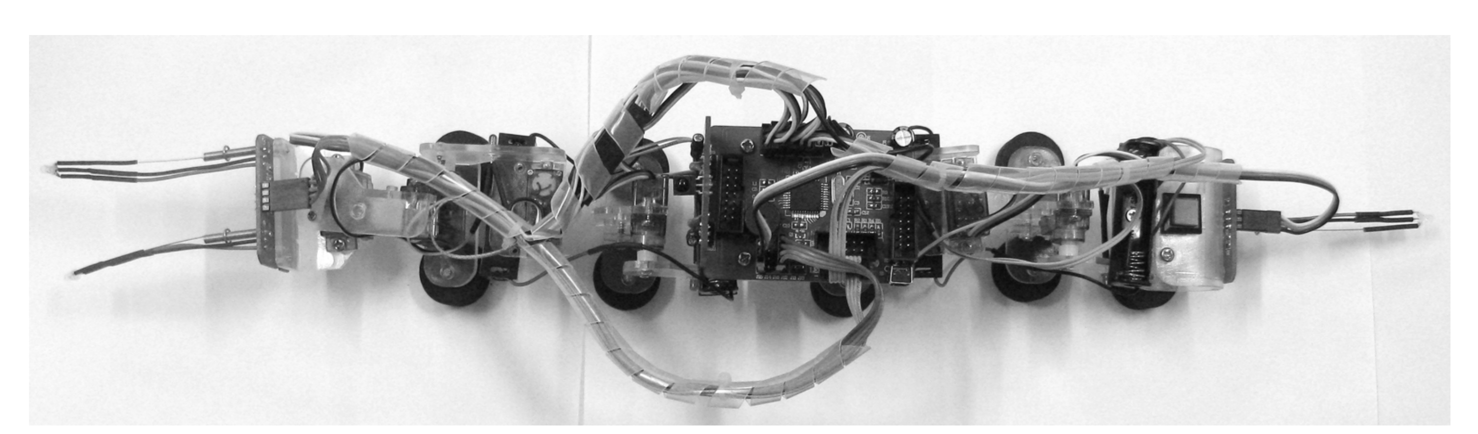
Schließen Sie die Spannungskonnektor noch nicht an!



Nachdem alles verdrahtet wurde, sieht Ihr Caterpillar wie folgt aus. Wir müssen die Kabel noch anordnen, lassen Sie aber noch genügend Platz übrig, damit der Caterpillar sich frei bewegen kann.



Binden Sie die Verkabelung wie skizziert zusammen. Binden Sie die Kabel nicht zu fest, damit Sie die Drähte noch leicht verschieben können.
Drehen Sie den Caterpillar vorsichtig in alle Richtungen.
Prüfen Sie dabei, ob keine Drähte zu straff gezogen werden. Führen Sie die Kabel beidseitig des Rollsensors gleichmäßig verteilt.



Benutzen Sie die Spirale, um die Kabel sauber zu verlegen. Seien Sie vorsichtig mit den LED-Anschlüssen und der Batterieverdrahtung. Wenn diese nicht locker genug geführt werden, führt das zu Metallermüdung und Kabelbruch bei einem sich stetig bewegenden Caterpillar.

Bauphase 29b: Prüfen der Batteriespannung

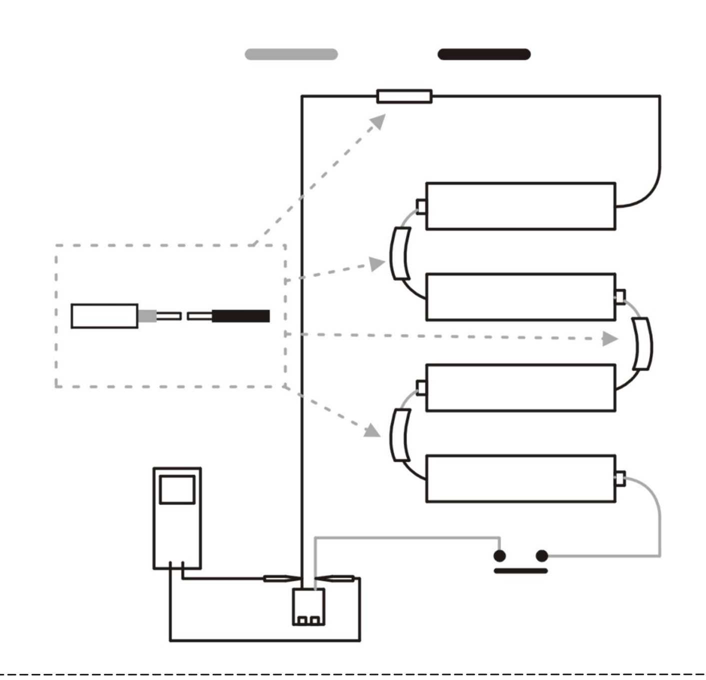
ACHTUNG!

SPANNUNG AM VERBINDER

Prüfen Sie erst die Batteriespannung am Verbinder, bevor Sie die Platine unter Spannung setzen!

Mit 6 Volt (Batterien) Nicht vergessen: Stecker J11 öffnen!

Mit 4,8 Volt (Akkus) Nicht vergessen: Stecker J11 geschlossen!



Bauphase 29b: Prüfen der Kabelverbindungen

ACHTUNG!

Prüfen Sie erst alle Kabelverbindungen an der Platine!

Vergessen Sie nicht, Stecker J11 bei normalen 1,5 Volt Batterien zu entfernen!

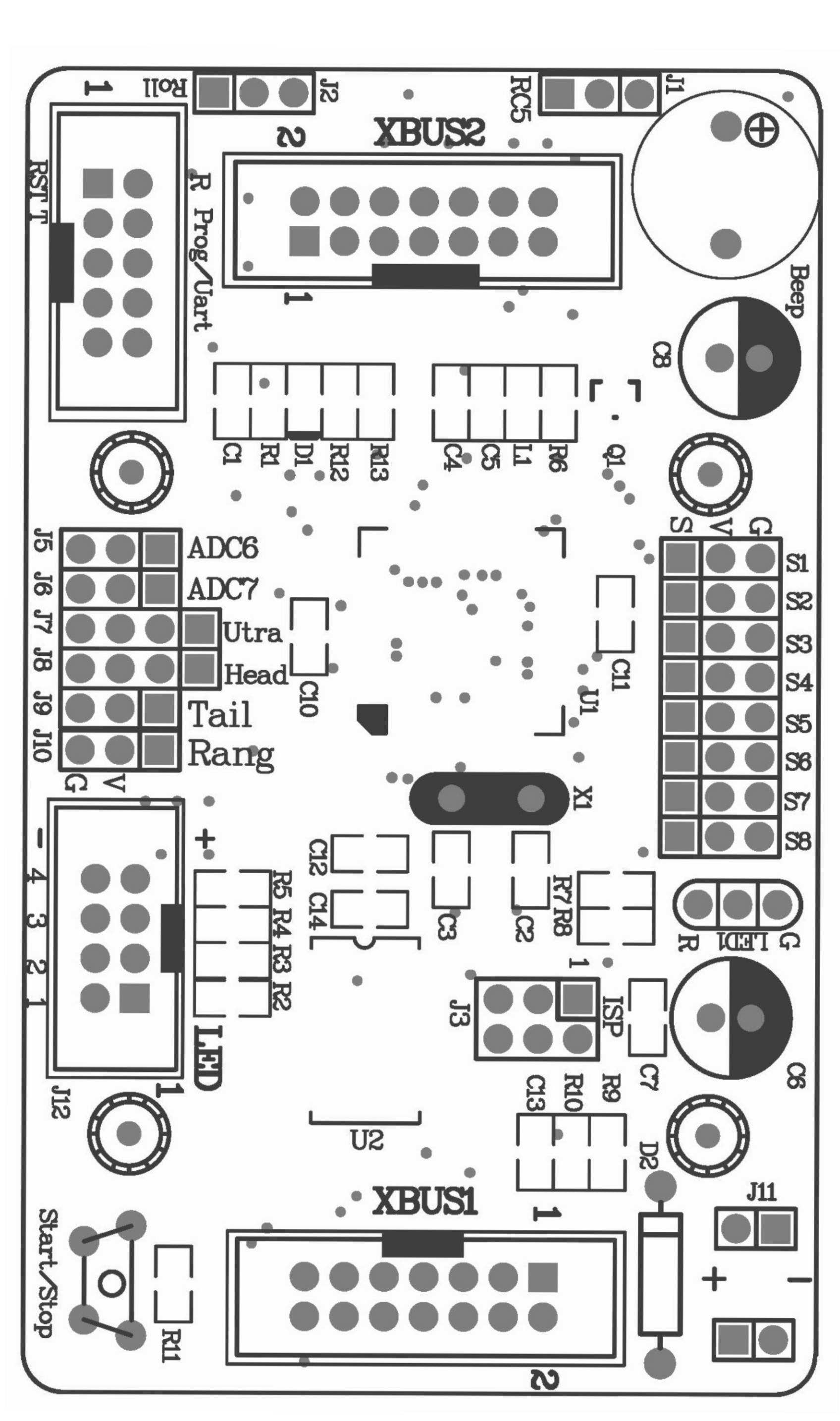
Seite 20; Software Installation

Seite 31; Erster Test

Seite 34; Selbsttest

ACHTUNG

Die maximale Spannung welche der RobotLoader messen kan ist 5,1 Volt!



4. Software Installation

Als nächstes kommen wir zur Software Installation. Die korrekt installierte Software wird für alle nachfolgenden Kapitel unbedingt benötigt.

Es sind Administrator-Rechte erforderlich, also melden Sie sich ggf. vorher als Administrator in Ihrem System an!

Wir empfehlen Ihnen erstmal das gesamte Kapitel in Ruhe durchzulesen und erst dann Schritt für Schritt die Installationsanleitung durchzugehen!

Grundlegende Kenntnis der Bedienung von Computern mit Windows oder Linux Betriebssystemen und den gängigen Programmen wie Dateimanager, Webbrowser, Texteditor, Packer (WinZip, WinRAR, unzip o.ä.) und ggf. Linux-Shell etc. **muss vorrausgesetzt werden!** Wenn Sie sich also nur wenig mit Computern auskennen, sollten Sie sich auf jeden Fall gut damit vertraut machen *bevor* Sie den Caterpillar in Betrieb nehmen! Eine Einführung in die Bedienung von Computern ist nicht Ziel dieser Anleitung und würde den Rahmen bei weitem sprengen! Hier geht es nur um den Caterpillar, dessen Programmierung und die speziell dafür benötigte Software.

Die Caterpillar CD-ROM
Sie haben vermutlich die Caterpillar CD-ROM im Laufwerk Ihres Computers – falls doch nicht, legen Sie diese nun bitte ein! Es sollte unter Windows kurz darauf per Autostart das CD Menü erscheinen. Andernfalls können Sie über einen Dateimanager die Datei "start.htm" im Hauptverzeichnis der CD mit einem Webbrowser wie z.B. Firefox öffnen. Die Installationsdateien für Firefox finden Sie übrigens auf der CD im Ordner

<CD-ROM-Laufwerk>:\Software\Firefox

sofern Sie noch keinen aktuellen Webbrowser installiert haben sollten. (Es sollte mindestens Firefox 1.x oder der Internet Explorer 6 sein...)

Nach Auswahl der Sprache finden Sie im CD Menü neben dieser Anleitung (die es auch zum Download auf unserer Homepage gibt), vielen Informationen, Datenblättern und Fotos auch den Menüpunkt "Software". Hier sind alle Software Tools, der USB Treiber und die Beispielprogramme mit Quellcode für den Caterpillar zu finden.

Abhängig von den Sicherheitseinstellungen Ihres Webbrowsers können Sie die Installationsprogramme direkt von der CD starten! Wenn Ihr Browser dies aufgrund der Sicherheitseinstellungen nicht erlaubt, müssen Sie die Dateien zunächst in ein Verzeichnis auf Ihrer Festplatte speichern und dann von dort starten. Genaueres dazu steht auf der Software Seite des CD Menüs. Alternativ können Sie natürlich auch direkt in einem Dateimanager auf das CD-Laufwerk wechseln und die Software von dort installieren. Die Verzeichnisnamen sind so gewählt, dass sie eindeutig den entsprechenden Softwarepaketen und Betriebssystemen zugeordnet werden können.

WinAVR - für Windows

Als erstes werden wir WinAVR installieren. *Win*AVR ist aber - wie der Name schon andeutet - nur für *Win* dows verfügbar !

Linux Anwender müssen beim nächsten Abschnitt weiterlesen.
WinAVR (das wird wie das englische Wort "whenever" ausgesprochen) ist eine Sammlung von vielen nützlichen und notwendigen Programmen für die Software Entwicklung für AVR Mikrocontroller in der Sprache C. WinAVR enthält neben dem GCC für AVR (das nennt sich dann insgesamt "AVR-GCC", mehr Infos dazu folgen später) auch den komfortablen Quelltexteditor "Programmers Notepad 2", den wir auch für die Programmentwicklung für den Caterpillar einsetzen werden!

WinAVR ist ein privat organisiertes Projekt, hinter dem keine Firma o.ä. steht - es ist kostenlos im Internet verfügbar. Neuere Versionen und weitere Informationen finden Sie hier:

http://winavr.sourceforge.net/

Inzwischen wird das Projekt aber auch offiziell von ATMEL unterstützt, und der AVRGCC lässt sich in AVRStudio, die Entwicklungsumgebung für AVRs von ATMEL, einbinden. Das werden wir in diesem Handbuch aber nicht beschreiben, für unsere Zwecke ist Programmers Notepad besser geeignet.
Die WinAVR Installationsdatei finden Sie auf der CD im Ordner:

<CD-ROM-Laufwerk>:\Software\AVR-GCC\Windows\WinAVR\

Die Installation von WinAVR ist sehr einfach und selbsterklärend - normalerweise brauchen keinerlei Einstellungen geändert werden – also einfach immer auf "Weiter" klicken!

Wenn Sie Windows Vista oder Windows 7 benutzen, müssen Sie auf jeden Fall die neueste Version von WinAVR verwenden! Auch mit Windows 2k und XP sollte es problemlos klappen. Falls nicht, können Sie eine der beiden älteren Versionen ausprobieren, die ebenfalls auf der CD zu finden sind (vor Neuinstallation immer bereits installierte WinAVR Versionen wieder deinstallieren!). Offiziell wird Win x64 noch nicht unterstützt, aber auf der CD findet sich ein Patch für Win x64 Systeme falls es Probleme geben sollte. Mehr Infos dazu finden Sie auf der Software Seite des CD Menüs!

AVR-GCC, avr-libc und avr-binutils - für Linux

Windows Anwender können diesen Abschnitt überspringen!

Unter Linux kann es schon ein wenig aufwändiger werden. Bei einigen Distributionen sind die benötigten Pakete zwar schon vorhanden, aber meist nur veraltete Versionen. Deshalb müssen Sie neuere Versionen kompilieren und einrichten. Wir können hier nicht im Detail auf jede der zahlreichen Linux Distributionen wie SuSE, Ubuntu, RedHat/Fedora, Debian, Gentoo, Slackware, Mandriva etc. pp. in zig verschiedenen Versionen mit ihren jeweiligen Eigenheiten eingehen und beschreiben das daher nur allgemein.

Das gilt auch für alle anderen Linux Abschnitte in diesem Kapitel!

Das hier beschriebene Vorgehen muss also bei Ihnen nicht unbedingt zum Erfolg führen. Oft kann es hilfreich sein, wenn Sie im Internet z.B. nach "<LinuxDistribution> avr gcc" o.ä. suchen (Verschiedene Schreibweisen ausprobieren). Auch das gilt für alle anderen Linux Abschnitte - natürlich mit angepassten Suchbegriffen! Falls Sie Probleme bei der Installation des AVR-GCC haben, können Sie auch mal in unserem oder im Roboternetz Forum nachschauen bzw. in einem der zahlreichen Linux Foren. Zunächst müssen Sie evtl. schon installierte Versionen des avr-gcc, der avr-binutils und der avr-libc deinstallieren – wie schon gesagt sind diese meist veraltet. Das können Sie mit dem jeweiligen Paketmanager ihrer Distribution tun indem Sie nach „avr"

Inbetriebnahme

suchen und die drei oben genannten Pakete deinstallieren – sofern diese überhaupt vorhanden sind. Ob der avr-gcc schon installiert ist oder nicht und wenn ja wo, können Sie über eine Konsole z.B. leicht mit

```
> which avr-gcc
```

herausfinden. Sollte hier ein Pfad angezeigt werden, ist schon eine Version installiert.
Geben Sie in diesem Fall einfach mal:

```
> avr-gcc --version
```

ein und schauen Sie sich die Ausgabe an. Sollte eine Versionsnummer kleiner als 3.4.6 angezeigt werden, müssen Sie diese alte Version auf jeden Fall deinstallieren.

Wenn die Versionsnummer zwischen 3.4.6 und 4.1.0 liegt, können Sie erstmal versuchen ob Sie Programme kompilieren können (s. nachfolgende Kapitel) und erst wenn das fehlschlägt, die neuen Tools installieren. Wir installieren im Folgenden die derzeit aktuelle Version 4.1.1 (Stand von März 2007) mit einigen wichtigen Patches.

Werden die oben genannten Pakete nicht im Paketmanager angezeigt, obwohl definitiv schon ein avr-gcc vorhanden ist, müssen Sie die entsprechenden Binärdateien manuell löschen – also die /bin, /usr/bin usw. Verzeichnisse nach allen Dateien, die mit „avr-" anfangen absuchen, und diese dann löschen (natürlich NUR diese Dateien und sonst nichts anderes!). Eventuell vorhandene Verzeichnisse wie /usr/avr oder /usr/local/ avr müssen ebenfalls gelöscht werden.

Achtung: Sie müssen unbedingt sicherstellen, dass die normalen Linux Entwicklungstools wie GCC, make, binutils, libc, etc. installiert sind, bevor Sie mit dem Übersetzen und der Installation beginnen können! Das tun Sie am besten über den Paketmanager Ihrer Distribution. Jede Linux Distribution sollte die benötigten Pakete schon auf der Installations CD mitliefern bzw. aktuelle Pakete über das Internet bereitstellen.

Stellen Sie sicher, dass das Programm „texinfo" installiert ist. Installieren Sie bitte ggf. das entsprechende Paket, bevor Sie weitermachen – sonst klappt es nicht!

Ist das erledigt, kann mit der eigentlichen Installation begonnen werden.

Es gibt nun zwei Möglichkeiten, entweder man macht alles von Hand, oder man nutzt ein sehr einfach anzuwendendes Installationsskript.

Wir empfehlen es zunächst mit dem Skript zu versuchen. Wenn das nicht klappt, kann man immer noch den Compiler von Hand einrichten!

Achtung: Sie sollten für die Installation noch genug freien Speicherplatz auf der Festplatte zur Verfügung haben! Temporär werden mehr als 400MB benötigt. Über 300MB davon können nach der Installation wieder gelöscht werden, aber während der Übersetzung braucht man den Platz.

Viele der nachfolgenden Installationsschritte erfordern **ROOT RECHTE,** also loggen Sie sich ggf. mit „su" als root ein oder führen Sie die kritischen Befehle mit „sudo" o.ä. aus, wie man es z.B. bei Ubuntu machen muss (das Installationsskript, mkdir in /usr/local Verzeichnissen und make install brauchen root Rechte).

Achten Sie im Folgenden bitte auf EXAKTE Schreibweise aller Befehle! Jedes Zeichen ist wichtig und auch wenn einige Befehle evtl. etwas seltsam aussehen – das ist alles richtig so und kein Tippfehler! (<CD-ROM-Laufwerk> muss man natürlich trotzdem mit dem Pfad des CD-ROM-Laufwerks ersetzen!)

Alle für uns relevanten Installationsdateien für den avr-gcc, avr-libc und binutils finden Sie auf der CD im Ordner:

<CD-ROM-Laufwerk>:\Software\avr-gcc\Linux

Zunächst müssen Sie alle Installationsdateien in ein Verzeichnis auf Ihrer Festplatte kopieren – **das gilt für beide Installationsvarianten!** Hier nutzen wir das Home Verzeichnis (übliche Abkürzung für das aktuelle Home Verzeichnis ist die Tilde: „~"):

```
> mkdir ~/Caterpillar
> cd <CD-ROM-Laufwerk>/Software/avr-gcc/Linux
> cp * ~/Caterpillar
```

Die Dateien können Sie nach der erfolgreichen Installation natürlich wieder löschen um Platz zu sparen!

Automatisches Installationsskript

Wenn man das Skript mit chmod ausführbar gemacht hat, kann es sofort losgehen:

```
> cd ~/Caterpillar
> chmod -x avrgcc_build_and_install.sh
> ./avrgcc_build_and_install.sh
```

Die Nachfrage, ob man mit dieser Konfiguration installieren möchte oder nicht, können Sie mit „y" beantworten.

ACHTUNG: Das Übersetzen und Installieren wird dann je nach Rechenleistung Ihres Systems einige Zeit in Anspruch nehmen. (z.B. etwa 15 min auf einem 2GHz CoreDuo Notebook – bei langsameren Systemen evtl. entsprechend länger)

Das Skript spielt auch einige Patches ein – das sind diese ganzen .diff Dateien, die in dem Verzeichnis liegen.

Wenn alles klappt, sollte ganz zum Schluss folgendes erscheinen:

```
(./avrgcc_build_and_install.sh)
(./avrgcc_build_and_install.sh) installation of avr GNU tools complete
(./avrgcc_build_and_install.sh) add /usr/local/avr/bin to your path to use the avr GNU tools
(./avrgcc_build_and_install.sh) you might want to run the following to save disk space:
(./avrgcc_build_and_install.sh)
(./avrgcc_build_and_install.sh) rm -rf /usr/local/avr/source /usr/local/avr/build
```

Dann können Sie wie es dort vorgeschlagen wird
rm -rf /usr/local/avr/source /usr/local/avr/build
ausführen! Das löscht alle temporären Dateien, die Sie normalerweise nicht mehr benötigen.

Jetzt können Sie den nächsten Abschnitt überspringen und noch den Pfad auf die avr tools setzen.

Sollte die Ausführung des Skriptes fehlschlagen, müssen Sie sich genau die Fehlermeldungen ansehen (auch mal in der Konsole etwas hochscrollen) – meist fehlen dann irgendwelche Programme, die man vorher noch installieren muss (wie z.B. das oben erwähnte texinfo). Bevor Sie nach einem Fehler weitermachen, sollten Sie die bereits erzeugten Dateien im Standardinstallationsverzeichnis „/usr/local/avr" vorsichtshalber löschen – am besten das ganze Verzeichnis.

Wenn Sie nicht wissen, was da genau falsch gelaufen ist, bitte alle Kommandozeilenausgaben in einer Datei speichern und damit an den Support wenden. Bitte immer so viele Informationen wie möglich mitsenden! Dann wird es einfacher, Ihnen zu helfen.

GCC für den AVR

Der GCC wird ähnlich wie die Binutils gepatcht, übersetzt und installiert:

```
> cd ~/Caterpillar
> bunzip2 -c gcc-4.1.1.tar.bz2 | tar xf -
> cd gcc-4.1.1
> patch -p0 < ../gcc-patch-0b-constants.diff
> patch -p0 < ../gcc-patch-attribute_alias.diff
> patch -p0 < ../gcc-patch-bug25672.diff
> patch -p0 < ../gcc-patch-dwarf.diff
> patch -p0 < ../gcc-patch-libiberty-Makefile.in.diff
> patch -p0 < ../gcc-patch-newdevices.diff
> patch -p0 < ../gcc-patch-zz-atmega256x.diff
> mkdir obj-avr
> cd obj-avr
> ../configure --prefix=$PREFIX --target=avr --enable-languages=c,c++ \
--disable-nls --disable-libssp –with-dwarf2
> make
> make install
```

Nach dem \ kann man einfach Enter drücken und weiterschreiben – so kann der Befehl auf mehrere Zeilen aufgeteilt werden. Kann man aber auch ganz weglassen.

AVR Libc

Und schließlich noch die AVR libc:

```
> cd ~/Caterpillar
> bunzip2 -c avr-libc-1.4.5.tar.bz2 | tar xf -
> cd avr-libc-1.4.5
> ./configure --prefix=$PREFIX --build=`./config.guess` --host=avr
> make
> make install
```

Achtung: bei –build=`./config.guess` darauf achten auch den „Accent grave" (à <-- den Strich da auf dem a! Neben der Backspace Taste – rechts oben auf der Tastatur, einmal mit Shift diese Taste drücken und danach die Leertaste) und kein normales Hochkomma oder Anführungszeichen zu benutzen, sonst klappt es nicht.

Pfad setzen

Sie müssen jetzt dafür sorgen, dass das Verzeichnis /usr/local/avr/bin auch in der Pfad Variablen eingetragen ist – sonst kann man den avr-gcc nicht aus der Konsole bzw. aus Makefiles heraus aufrufen. Dazu müssen Sie den Pfad in die Datei /etc/profile bzw. /etc/environment o.ä. (variiert von Distribution zu Distribution) eintragen – mit einem Doppelpunkt „:" getrennt von den anderen schon vorhandenen Einträgen. In der Datei könnte das dann *in etwa* so aussehen:

PATH="/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin**:/usr/local/avr/bin**"

Jetzt in einer beliebigen Konsole „avr-gcc -–version" eingeben wie weiter oben beschrieben – wenn das funktioniert, ist die Installation gelungen!

Manuelle Installation

Wenn Sie den Compiler lieber von Hand einrichten wollen oder die Installation mit dem Skript nicht klappt, können Sie nach den Anweisungen im folgenden Abschnitt vorgehen.

Die Beschreibung hier orientiert sich an diesem Artikel:

http://www.nongnu.org/avr-libc/user-manual/install_tools.html

der auch in der AVR Libc Dokumentation im PDF Format auf der CD zu finden ist:

<CD-ROM-Laufwerk>:\Software\Documentation\avr-libc-user-manual-1.4.5.pdf

ab PDF Seite 240 (bzw. 232 nach der Seitenzahl des Dokuments).

Wir fassen uns hier zwar sehr viel kürzer, spielen aber gleich noch ein paar wichtige Patches ein – ohne diese funktionieren einige Dinge nicht richtig.

Zunächst müssen wir uns ein Verzeichnis erstellen, in das wir alle Tools installieren werden. Das sollte /usr/local/avr sein.
Also in einer Konsole **ALS ROOT** folgendes eingeben:

```
> mkdir /usr/local/avr
> mkdir /usr/local/avr/bin
```

Es muss nicht unbedingt dieses Verzeichnis sein. Wir legen dafür einfach die Variable $PREFIX für dieses Verzeichnis an:

```
> PREFIX=/usr/local/avr
> export PREFIX
```

Das muss nun noch unbedingt der PATH Variable hinzugefügt werden:

```
> PATH=$PATH:$PREFIX/bin
> export PATH
```

Binutils für AVR

Nun müssen Sie den Quellcode der Binutils entpacken und ein paar Patches einspielen.
Wir nehmen hier an, dass Sie alles ins Home Verzeichnis ~/Caterpillar kopiert haben:

```
> cd ~/Caterpillar
> bunzip2 -c binutils-2.17.tar.bz2 | tar xf -
> cd binutils-2.17
> patch -p0 < ../binutils-patch-aa.diff
> patch -p0 < ../binutils-patch-atmega256x.diff
> patch -p0 < ../binutils-patch-coff-avr.diff
> patch -p0 < ../binutils-patch-newdevices.diff
> patch -p0 < ../binutils-patch-avr-size.diff
> mkdir obj-avr
> cd obj-avr
```

Nun wird das configure Skript ausgeführt:

```
> ../configure --prefix=$PREFIX --target=avr --disable-nls
```

Dieses Skript ermittelt, was auf Ihrem System verfügbar ist und erzeugt dementsprechend passende Makefiles. Jetzt können die Binutils übersetzt und installiert werden:

```
> make
> make install
```

Das kann je nach Rechenleistung Ihres Systems schon ein paar Minuten dauern – das gilt auch für die beiden nächsten Abschnitte – vor allem für den GCC!

Java 6

Der RobotLoader (Infos dazu s.u.) wurde für die Java Plattform entwickelt und ist unter Windows und Linux verwendbar (theoretisch auch andere Betriebsysteme wie OS X, aber hier kann AREXX Engineering leider noch keinen offiziellen Support leisten). Damit das funktioniert, ist es notwendig, ein aktuelles Java Runtime Environment (JRE) zu installieren. Oft haben Sie dies bereits auf dem Rechner, allerdings muss es mindestens Version 1.6 (= Java 6) sein! Falls Sie also noch kein JRE oder JDK installiert haben, müssen Sie zunächst das auf der CD mitgelieferte JRE 1.6 der Firma SUN Microsystems installieren, oder alternativ eine neuere Version von http://www.java.com oder http://java.sun.com downloaden.

Windows

Das JRE 1.6 befindet sich für Windows in folgendem Ordner:

<CD-ROM-Laufwerk>:\Software\Java\JRE6\Windows\

Unter Windows ist die Installation von Java sehr einfach - Sie müssen nur den Setup starten und den Anweisungen auf dem Bildschirm folgen - fertig. Den nächsten Abschnitt können Sie überspringen.

Linux
Unter Linux ist die Installation meistens auch relativ problemlos möglich, bei einigen Distributionen kann es aber ein wenig Handarbeit erfordern.

In diesem Ordner:

<CD-ROM-Laufwerk>:\Software\Java\JRE6\

finden Sie das JRE1.6 als RPM (SuSE, RedHat etc.) und als selbstextrahierendes Archiv „.bin". Unter Linux ist es besser wenn Sie zunächst im Paketmanager Ihrer jeweiligen distribution nach Java Paketen suchen (Suchbegriffe z.B. „java", „sun", „jre", „java6" ...) und dann diese distributionseigenen Pakete verwenden und nicht die auf dieser CD-ROM! Achten Sie aber unbedingt darauf Java 6 (= 1.6) oder ggf. eine neuere Version zu installieren und keine ältere Version!

Unter Ubuntu oder Debian funktioniert das RPM Archiv nicht direkt – hier müssen Sie die Paketmanager Ihrer jeweiligen Distribution bemühen, um an ein passendes Installationspaket zu kommen. Das RPM sollte bei vielen anderen Distributionen wie RedHat/Fedora und SuSE aber problemlos funktionieren. Falls nicht, bleibt noch der Weg das JRE aus dem selbstextrahierenden Archiv (.bin) zu entpacken (z.B.nach /usr/lib/Java6) und dann manuell die Pfade zum JRE zu setzen (PATH undJAVA_HOME etc.).

Bitte beachten Sie hier auch die Installationsanweisungen von Sun – die ebenfalls im oben genannten Verzeichnis und auf der Java Website (s.o.) zu finden sind!

Ob Java korrekt installiert wurde, können Sie in einer Konsole überprüfen, indem Sie den Befehl „java -version" ausführen. Es sollte in etwa folgende Ausgabe erscheinen:

```
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode, sharing)
```

Steht dort etwas ganz anderes, haben Sie entweder die falsche Version installiert, oder auf Ihrem System ist noch eine andere Java VM installiert.

Robot Loader

Der Robot Loader wurde entwickelt, um komfortabel neue Programme in den Caterpillar und alle Erweiterungsmodule laden zu können (sofern diese über einen Mikrocontroller mit kompatiblem Bootloader verfügen). Weiterhin sind ein paar nützliche Zusatzfunktionen integriert, wie z.B. ein einfaches Terminalprogramm.

Den Robot Loader selbst braucht man nicht zu installieren – das Programm kann einfach irgendwo in einen neuen Ordner auf die Festplatte kopiert werden. Der Robot Loader befindet sich in einem Zip-Archiv auf der CD-ROM:

<CD-ROM-Laufwerk>:\Software\RobotLoader\RobotLoader.zip

Dieses müssen Sie nur irgendwo auf die Festplatte entpacken – z.B. in einen neuen Ordner C:\Programme\RobotLoader (o.ä.). In diesem Ordner finden Sie dann die Datei RobotLoader.exe und können sie mit einem Doppelklick starten.

Das eigentliche Robot Loader Programm liegt im Java Archive (JAR) RobotLoader_lib.jar. Dieses können Sie alternativ auch von der Kommandozeile aus starten.

Unter Windows:

```
java -Djava.library.path=".\lib" -jar RobotLoader_lib.jar
```

Linux:

```
java -Djava.library.path="./lib" -jar RobotLoader_lib.jar
```

Diese lange -D Option ist notwendig, damit die JVM auch alle verwendeten Bibliotheken finden kann. Unter Windows braucht man das aber nicht und kann einfach die .exe Datei zum Starten verwenden und für Linux gibt es ein Shell Skript „RobotLoader. sh“. Das Skript muss evtl. zunächst noch ausführbar gemacht werden (chmod -x ./RobotLoader.sh). Danach kann man es in einer Konsole mit „./RobotLoader.sh“ starten.

Es empfiehlt sich, eine Verknüpfung auf dem Desktop oder im Startmenü anzulegen, um den Robot Loader bequem starten zu können. Unter Windows geht das z.B. einfach indem man rechts auf die Datei RobotLoader.exe klickt und dann im Menü „Senden an“ auf „Desktop (Verknüpfung erstellen)“ klickt.

Caterpillar Library, Caterpillar CONTROL Library und Beispielprogramme

Die Caterpillar Library und die zugehörigen Beispielprogramme befinden sich in einem Zip-Archiv auf der CD:

<CD-ROM-Laufwerk>:\Software\CaterpillarExamples\CaterpillarExamples.zip

Sie können diese einfach direkt in ein Verzeichnis Ihrer Wahl auf die Festplatte entpacken. Am besten entpacken Sie die Beispielprogramme in einen Ordner auf einer Daten Partition. Oder in den „Eigene Dateien“ Ordner in einem Unterordner „caterpillar\Examples\“ bzw. unter Linux ins Home Verzeichnis. Das steht Ihnen aber völlig frei.

Die einzelnen Beispielprogramme werden noch später im Softwarekapitel besprochen!

5. Robot Loader

5.1. Anschluss des USB Interfaces – Windows

Linux Anwender können beim nächsten Abschnitt weiterlesen!

Zur Installation des USB Interfaces gibt es mehrere Möglichkeiten. Die einfachste Möglichkeit ist es, **den Treiber VOR dem ersten Anschließen des Geräts zu installieren.**
Auf der CD befindet sich ein Installationsprogramm für den Treiber.

Für **32 und 64 Bit Windows 7, XP, Vista, Server 2003 und 2000** Systeme:

<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win2k_XP\CDM_Setup.exe

*Für alte **Win98SE/Me** Systeme gibt es so ein komfortables Programm leider nicht. Hier muss ein älterer Treiber von Hand installiert werden nachdem man das Gerät angeschlossen hat (s.u.).*

Das CDM Installationsprogramm müssen Sie einfach nur ausführen – es gibt nur eine kurze Rückmeldung, sobald der Treiber installiert wurde, sonst passiert nichts weiter.

Dann können Sie das USB Interface an den PC anschließen. ***BITTE NOCH NICHT MIT DEM ROBOTER VERBINDEN!*** Einfach nur über das USB Kabel mit dem PC verbinden! Dabei sollten Sie darauf achten, die Platine des USB Interfaces nur am Rand oder am USB Stecker bzw. an der Kunststoffwanne des Programmiersteckers anzufassen (s. Sicherheitshinweise zu statischen Entladungen)! Sie sollten besser *keine* der Bauteile auf der Platine, Lötstellen oder die Kontakte des Wannensteckers berühren, wenn nicht unbedingt nötig, um statische Entladungen zu vermeiden!

Der zuvor installierte Treiber wird nun automatisch für das Gerät verwendet, ohne dass Sie noch etwas zu tun brauchen. Es erscheinen bei Windows XP/2k kleine Sprechblasen unten über der Taskleiste – die letzte Meldung sollte in etwa „Die Hardware wurde installiert und kann nun verwendet werden!" lauten!

Wenn Sie das USB Interface doch schon vor der Installation angeschlossen haben (oder Win98/Me benutzen) – auch nicht schlimm. Dann werden Sie von Windows nach einem Treiber gefragt. Auch diese Installationsvariante ist möglich, der Treiber befindet sich auch in entpackter Form auf der CD!

Wenn dies bei Ihnen der Fall ist, erscheint (unter Windows) für gewöhnlich ein Dialog zum Installieren eines neuen Gerätetreibers. Sie müssen dem System dann den Pfad angeben, unter dem es den Treiber finden kann. Bei Windows 2k/XP muss man erst auswählen, den Treiber manuell zu installieren und natürlich keinen Webdienst o.ä. zu suchen. Der Treiber befindet sich in unserem Fall auf der CD in den oben genannten Verzeichnissen.

Also einfach das jeweilige Verzeichnis für Ihre Windows Version angeben und evtl. noch ein paar Dateien, die das System nicht selbstständig findet (sind alle in den weiter unten genannten Verzeichnissen!)...

Bei Windows XP oder späteren Versionen folgt oft (hier normalerweise nicht, da die FTDI Treiber signiert sind) noch ein Hinweis das der Treiber nicht von Microsoft signiert/verifiziert worden ist - das ist irrelevant und kann bedenkenlos bestätigt werden.

Inbetriebnahme

Für **32 und 64 Bit Windows 7, XP, Vista, Server 2003 und 2000** Systeme:

<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win2k_XP\FTDI_CDM2\

Für ältere **Windows 98SE/Me** Systeme:

<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win98SE_ME\FTDI_D2XX\

Bei einigen älteren Windows Versionen wie Win98SE ist evtl. nach Installation des Treibers ein Neustart erforderlich! **ACHTUNG:** Unter **Win98/Me** funktioniert nur einer von beiden Treibern: Virtual Comport oder der D2XX Treiber von FTDI! Hier gibt es leider keinen Treiber, der beide Funktionen integriert und es steht normalerweise kein virtueller Comport zur Verfügung, da der RP6Loader unter Windows standardmäßig die D2XX Treiber verwendet (das kann man auch ändern - kontaktieren Sie hier ggf.unser Support Team!).

Überprüfen, ob das Gerät richtig angeschlossen ist

Um zu überprüfen ob das Gerät korrekt installiert worden ist, kann man unter Windows XP, 2003 und 2000 neben dem Robot Loader auch den Gerätemanager verwenden:
Rechtsklick auf den Arbeitsplatz --> Eigenschaften --> Hardware --> Gerätemanager

ODER alternativ: Start --> Einstellungen --> Systemsteuerung --> Leistung und Wartung --> System --> Hardware --> Gerätemanager und dort in der Baumansicht unter "Anschlüsse (COM und LPT)" nachsehen ob ein "USB-Serial Port (COMX)" zu sehen ist - wobei das X für die Portnummer steht oder unter „USB-Controller“ nach einem „USB Serial Converter“ suchen!

Treiber später wieder Deinstallieren

Sollten Sie den Treiber jemals wieder deinstallieren wollen *(Nein, das tun Sie jetzt bitte nicht - ist nur ein Hinweis falls Sie das jemals brauchen sollten)*: Wenn Sie das CDM Installationsprogramm verwendet haben, können Sie das direkt über Start-->Einstellungen--> Systemsteuerung-->Software tun. In der dortigen Liste finden Sie einen Eintrag des „FTDI USB Serial Converter Drivers“ – diesen auswählen und dort dann auf deinstallieren klicken!

Wenn Sie den Treiber von Hand installiert haben, können Sie das Programm "FTUNIN.exe" im Verzeichnis des jeweiligen USB Treibers für Ihr System ausführen!
Achtung: USB-->RS232 Adapter mit FTDI Chipsatz verwenden meist ebenfalls diesen Treiber!

5.2. Anschluss des USB Interfaces – Linux

Windows Anwender können diesen Abschnitt überspringen!

Bei Linux mit Kernel 2.4.20 oder höher ist der benötigte Treiber schon vorhanden (zumindest für das kompatible Vorgängermodell FT232BM des Chips auf unserem USB Interface, dem FT232R), das Gerät wird automatisch erkannt und Sie brauchen nichts weiter zu tun. Falls es doch mal Probleme gibt, erhalten Sie Linux Treiber (und Support und auch evtl. neuere Treiber) direkt von FTDI:

http://www.ftdichip.com/

Unter Linux kann man, nachdem man das Gerät angeschlossen hat, mit:

cat /proc/tty/driver/usbserial

anzeigen lassen, ob der USB-Serial Port korrekt installiert worden ist. Mehr braucht man hier normalerweise nicht zu tun.

Allerdings sei noch darauf hingewiesen, dass der Robot Loader unter Windows die D2XX Treiber verwendet und dort die vollständigen USB Bezeichnungen in der Portliste auftauchen (z.B. „USB0 | Robot USB Interface | serialNumber“). Unter Linux sind es stattdessen die virtuellen Comport Bezeichnungen /dev/ttyUSB0, /dev/ttyUSB1 etc.. Es werden auch die normalen Comports angezeigt, als „dev/ttyS0“ usw.. Hier müssen Sie ausprobieren welcher Port der richtige ist!

Für Linux ist leider kein so einfach zu installierender Treiber verfügbar der beides bereitstellt. Von daher war es hier sinnvoller die Virtual Comport Treiber, die ohnehin schon im Kernel vorhanden sind, zu verwenden. Die D2XX Treiber würden bei der Installation auch noch einiges an Handarbeit erfordern...

Software Installation abschließen

Das war es auch schon mit der Installation der Software und des USB Interfaces! Jetzt könnten Sie noch die wichtigsten Dateien von der CD auf die Festplatte kopieren (vor allem den kompletten Ordner „Documentation“ und, falls nicht schon geschehen, die Beispielprogramme). Dann müssen Sie nicht ständig die CD suchen, wenn Sie diese Dateien benötigen! Die Ordner auf der CD sind alle so benannt, dass sie eindeutig den jeweiligen Softwarepaketen bzw. der jew. Dokumentation zugeordnet werden können!

Sollten Sie die CD einmal "verlegen", können Sie die wichtigsten Dateien wie dieses Handbuch, den Robot Loader und die Beispielprogramme auch von der AREXX Homepage downloaden. Dort finden Sie auch Links zu den anderen Softwarepaketen, die Sie benötigen.

5.3. Erster Test

Stellen Sie nun bitte unbedingt sicher, dass der Schalter auf dem Roboter in der Position OFF steht!

Mit Normale 1,5 Volt Batterien Stecker J19 entfernen!
Mit Akkus Stecker J11 installieren!

Jetzt können Sie 4 NiMH Mignon Akkus oder normale 1,5 Volt Batterien ***POLUNGSRICHTIG*** in den Akkuhalter einlegen!

Vorsicht: Wenn Sie die Batterien oder Akkus falsch herum einlegen, wird die Elektronik normalerweise durchbrennen! ***Kontrollieren Sie lieber dreimal, ob die Batterien oder Akkus auch wirklich richtig herum in den Halter eingesetzt sind!***
Prüfen Sie jetzt erst die BATTERIESPANNUNG am Batteriekontakt.

ACHTUNG! Lesen Sie diesen und den folgenden Abschnitt komplett durch, bevor Sie den Roboter anschalten! Sollte dann etwas nicht so ablaufen wie hier beschrieben, schalten Sie den Roboter am besten *sofort* aus und notieren Sie sich genau, was falsch gelaufen ist! Wenn Sie für den Fehler keine Lösung gefunden haben, wenden Sie sich bitte an den Support in unserem Forum!

Jetzt kann es los gehen - der Roboter wird nun das erste Mal eingeschaltet! Schalten Sie den Roboter am Schwanzhauptschalter ein. Es sollten nun zwei Kopf LEDs und die Schwanz LED Grün aufleuchten.

Die Power LED (LED1) sollte für etwa 20 Sekunde nach dem Anschalten ROT Blinken (Ohne Program in ATMEGA16) oder Grün Blinken (Mit Program in ATMEGA16) und dann wieder ausgehen.

Wann die Spannung < 4,4 Volt Blinkt die Power LED, ROT oder Grün/ORANGE und blinken auch die 4 Rote LEDs auf Caterpillar.

ACHTUNG

Die maximale Spannung welche der RobotLoader messen kan ist 5,1 Volt!

5.4. USB Interface anschließen und RobotLoader starten

Als nächstes testen wir den Programmupload über das USB Interface. Verbinden Sie bitte das USB Interface mit dem PC **(Immer zuerst mit dem PC verbinden!)** und *danach* über das 10-pol. Flachbandkabel mit dem "PROG/UART" Anschluss des Caterpillars! **(Caterpillar MUSS AUS stehen!)** Das 10-pol. Flachbandkabel ist mechanisch gegen Verpolung geschützt, sofern man es also nicht mit Gewalt behandelt, kann man es gar nicht verkehrt herum anschließen.

Starten Sie danach den RobotLoader. Je nachdem welche Sprache Sie gewählt haben, können die Menüs natürlich etwas anders beschriftet sein. Auf den Screenshots ist die englische Sprachversion dargestellt, über den Menüpunkt „Options->Preferences" und dann bei „Language / Sprache" kann man die Sprache anpassen (Englisch oder Deutsch) und danach auf OK klicken.Nach Änderung der Sprache muss man den RobotLoader aber erst einmal neu starten bevor sich etwas ändert!

Port öffnen - Windows

Jetzt können Sie den USB Port auswählen. Sofern kein anderer USB->Seriell Adapter mit FTDI Controller am PC angeschlossen ist, sehen sie in der Portliste nur einen einzigen Eintrag den Sie dann bitte auswählen. Falls doch mehrere Ports vorhanden sind, können Sie den Port anhand des Namens „Robot USB Interface" identifizieren (oder „FT232R USB UART"). Dahinter wird noch die einprogrammierte Seriennummer angezeigt.

Sollten keine Ports angezeigt werden, können sie im Menü über „RobotLoader-->Refresh Portlist" („RobotLoader-->Portliste aktualisieren") die Portliste aktualisieren!

Port öffnen – Linux

Unter Linux wird der USB-Seriell Adapter wie ein normaler Comport behandelt. Die D2XX Treiberinstallation von FTDI unter Linux wäre nicht ganz so einfach und die normalen Virtual Comport (VCP) Treiber sind in aktuellen Linux Kerneln sowieso schon enthalten. Es funktioniert alles fast genauso wie unter Windows, nur muss man erst noch kurz ausprobieren welchen Namen das Caterpillar USB Interface hat und darauf achten, den USB Port nicht vom PC zu trennen solange die Verbindung noch offen ist (ansonsten muss der RobotLoader eventuell neu gestartet werden damit die Verbindung wieder klappt). Die Virtual Comports heissen unter Linux „/dev/ttyUSBx", wobei x eine Nummer ist, z.B. „/dev/ttyUSB0" oder „/dev/ttyUSB1". Die normalen Comports heissen unter Linux „/dev/ttyS0", „/dev/tty- S1" etc.. Diese tauchen ebenfalls in der Portliste auf sofern vorhanden.

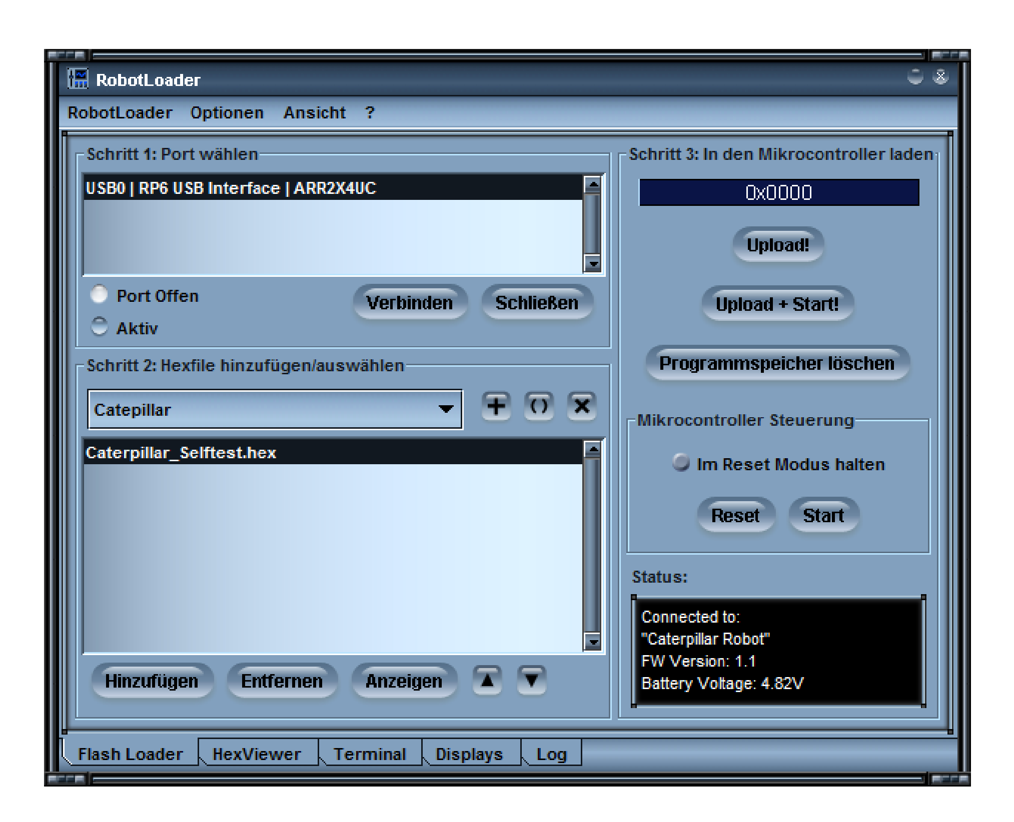
Der RobotLoader merkt sich – wenn es denn überhaupt mehrere Ports gibt - welchen Port Sie zuletzt verwendet haben und selektiert diesen bei jedem Start des Programms automatisch (generell bleiben die meisten Einstellungen und Selektionen erhalten).

Nun können Sie auf den Button „Connect“ (=„Verbinden“) klicken! Der RobotLoader öffnet nun den Port und testet, ob die Kommunikation mit dem Bootloader auf dem Roboter funktioniert. Es sollte dann unten im schwarzen Feld „Status“ die Nachricht "Connected to: Caterpillar Robot ..." o.ä. zusammen mit einer Info über die aktuell gemessene Akkuspannung erscheinen. Falls nicht, probieren Sie es nochmal! Wenn es dann immer noch nicht klappt, liegt ein Fehler vor! Schalten Sie in diesem Fall den Roboter am Besten sofort aus und fangen Sie an mit Fehlersuche!

Ist die Spannung zu niedrig, erscheint eine Warnung. **Spätestens jetzt sollten die Akkus neu geladen werden** (besser schon dann, wenn die Spannung unter etwa 4,0V gefallen ist)!

Wenn das geklappt hat, kann ein kleines Selbsttestprogramm ausgeführt werden, um die Funktionsfähigkeit aller Systeme des Roboters zu überprüfen. Klicken Sie dazu unten im Robot Loader Fenster auf den Button „Add“ („Hinzufügen“) und wählen Sie die Datei CaterpillarExamples, „Example_11_Selftest\Caterpillar_SELFTEST.hex“ im Beispielverzeichnis aus. In dieser Datei befindet sich das Selbsttestprogramm im hexadezimalen Format – daher werden solche Programmdateien auch „Hexdateien“ genannt. Die eben ausgewählte Datei taucht anschließend in der Liste auf. So können Sie später auch noch andere Hexdateien von Ihren eigenen Programmen und den Beispielprogrammen hinzufügen (s. Screenshot, hier wurden schon ein paar Hexdateien hinzugefügt). Der Robot Loader kann auch verschiedene Kategorien von Hexdateien verwalten. Damit lassen sich die Dateien übersichtlicher sortieren. Beispielsweise wenn man mehrere programmierbare Erweiterungsmodule auf dem Caterpillar hat, oder verschiedene Varianten von Programmen verwendet. Die Liste wird immer automatisch beim Beenden des Programms gespeichert! Es werden hier natürlich nur die Pfade zu den Hexdateien gespeichert – nicht die Hexdateien selbst. Wenn Sie an einem Programm arbeiten, brauchen Sie die Hexdatei nur einmal hinzuzufügen und auszuwählen, und können danach sofort nach jedem erneuten Übersetzen des Programms, das neue Programm in den Mikrocontroller laden (auch per Tastenkombination [STRG+D] oder [STRG+Y], um das Programm direkt nach dem Übertragen zu starten). Unter verschiedenen Betriebssystemem sind die Pfadnamen natürlich komplett anders. Sie können den Robot Loader trotzdem ohne weitere Änderungen direkt unter Windows und Linux verwenden, denn es gibt für Windows und Linux jeweils eine extra Liste.

Wählen Sie jetzt die Datei „Caterpillar_Selftest.hex“ in der Liste aus und klicken Sie dann auf den Button „Upload!“ oben rechts unter dem Fortschrittsbalken. Das Programm wird nun in den MEGA16 auf dem Caterpillar hochgeladen. Das sollte nicht länger als ein paar Sekunden dauern (maximal 5 Sekunden beim Selbsttest Programm).

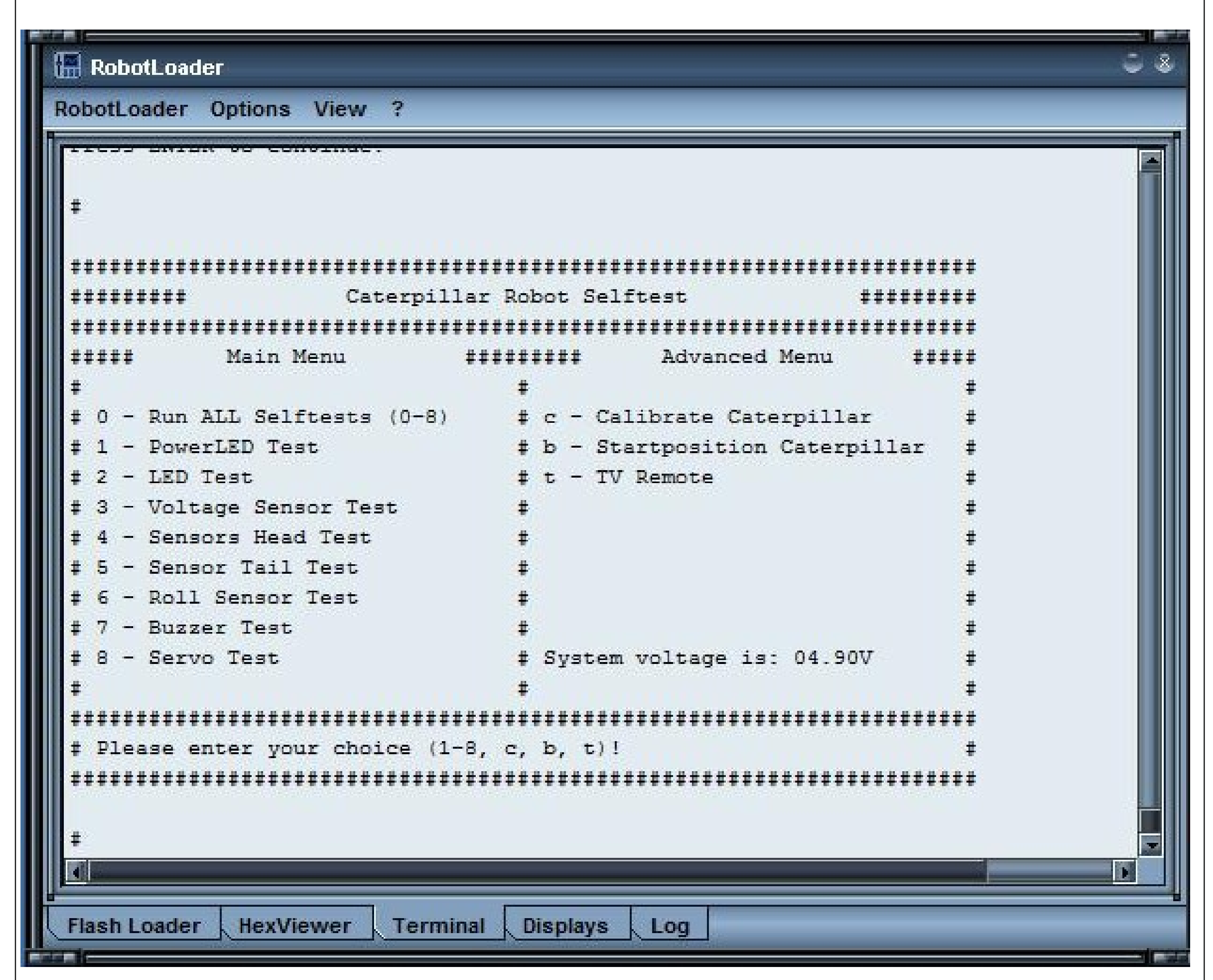


Jetz konnen Sie weiter gehn mit die Selftest.

5.5. Selbsttest

Wann ein Programm im Caterpillar geladen ist Blinkt die Power LED Grün bei ein Spannung > 4,4 Volt. Ist die Akkuspannung zu niedrig, erscheint eine Warnung und Blinkt die Power LED Grün/Orange und Blinken die 4 Rotte LEDs.

Wechseln Sie anschließend bitte auf den Karteireiter (=„Tabs“ ganz unten im Programmfenster!) „Terminal“! Alternativ geht das auch über das Menü „View“ („Ansicht“).



Sie können jetzt den Selbsttest und die Kalibrierung des Caterpillar vornehmen.
Starten Sie dann das Programm mit einem Druck auf den Start/Stop Taster auf dem Caterpillar, neben dem LEDanschluss! Später können Sie das natürlich auch alternativ über das Menü RobotLoader --> Start oder die Tastenkombination [STRG]+[S] tun, so können Sie allerdings direkt ausprobieren ob der Taster korrekt funktioniert!

Falls im Selbsttest ein Fehler auftritt, schalten Sie den Roboter am Besten sofort aus und fangen Sie mit der Fehlersuche an.

Wenn der Selbsttest OK ist, können Sie den Caterpillar kalibrieren (C - Calibrate). Das Ergebnis der Kalibrierung wird im ATMEGA Prozessor des Caterpillar gespeichert und ist Ihre Startposition für die weitere Programmierung.

Sie können nun entweder mit den anderen Beispielprogrammen (Examples) des Caterpillar weitermachen oder mit Ihrer eigenen Softwareprogrammierung anfangen.

6. Programmierung des CATERPILLAR

Nun kommen wir so langsam zur Programmierung des Roboters.

Einrichten des Quelltexteditors

Erstmal müssen wir uns eine kleine Entwicklungsumgebung einrichten. Der sog. „Quelltext“ (auch Quellcode oder engl. Sourcecode genannt) für unsere C Programme muss ja irgendwie in den Computer eingegeben werden!

Dazu werden wir natürlich auf gar keinen Fall Programme wie OpenOffice oder Word verwenden! Vielleicht ist das nicht für jeden offensichtlich, deshalb wird es hier explizit betont. Damit kann man zwar gut Handbücher wie dieses hier schreiben, aber zum Programmieren ist das absolut ungeeignet. Quelltext ist reiner Text – ohne jegliche Formatierung. Schriftgröße oder Farbe interessieren den Compiler nicht...

Für einen Menschen ist es natürlich übersichtlicher, wenn bestimmte Schlüsselwörter oder Arten von Text automatisch farbig hervorgehoben werden. Das und noch einiges mehr bietet Programmers Notepad 2 (im folgenden kurz „PN2“ genannt), der Quelltexteditor, den wir verwenden werden *(ACHTUNG: Unter Linux müssen Sie einen anderen Editor verwenden, der ähnliches wie PN2 bietet. Es sind für gewöhnlich mehrere bereits vorinstalliert! (z.B. kate, gedit, exmacs o.ä.))*. Neben dem Hervorheben von Schlüsselwörtern und ähnlichem (sog. „Syntax Highlighting“) gibt es auch eine rudimentäre Projektverwaltung. Man kann so mehrere Quelltextdateien in Projekten organisieren und in einer Liste alle zu einem Projekt gehörenden Dateien anzeigen lassen. Weiterhin kann man aus PN2 komfortabel Programme wie den AVR-GCC aufrufen und so die Programme bequem über einen Menüeintrag übersetzen lassen. Der AVR-GCC ist ja normalerweise ein reines Kommandozeilenprogramm ohne graphische Oberfläche...

Neuere Versionen von Programmers Notepad finden Sie auf der Projekthomepage:
http://www.pnotepad.org/

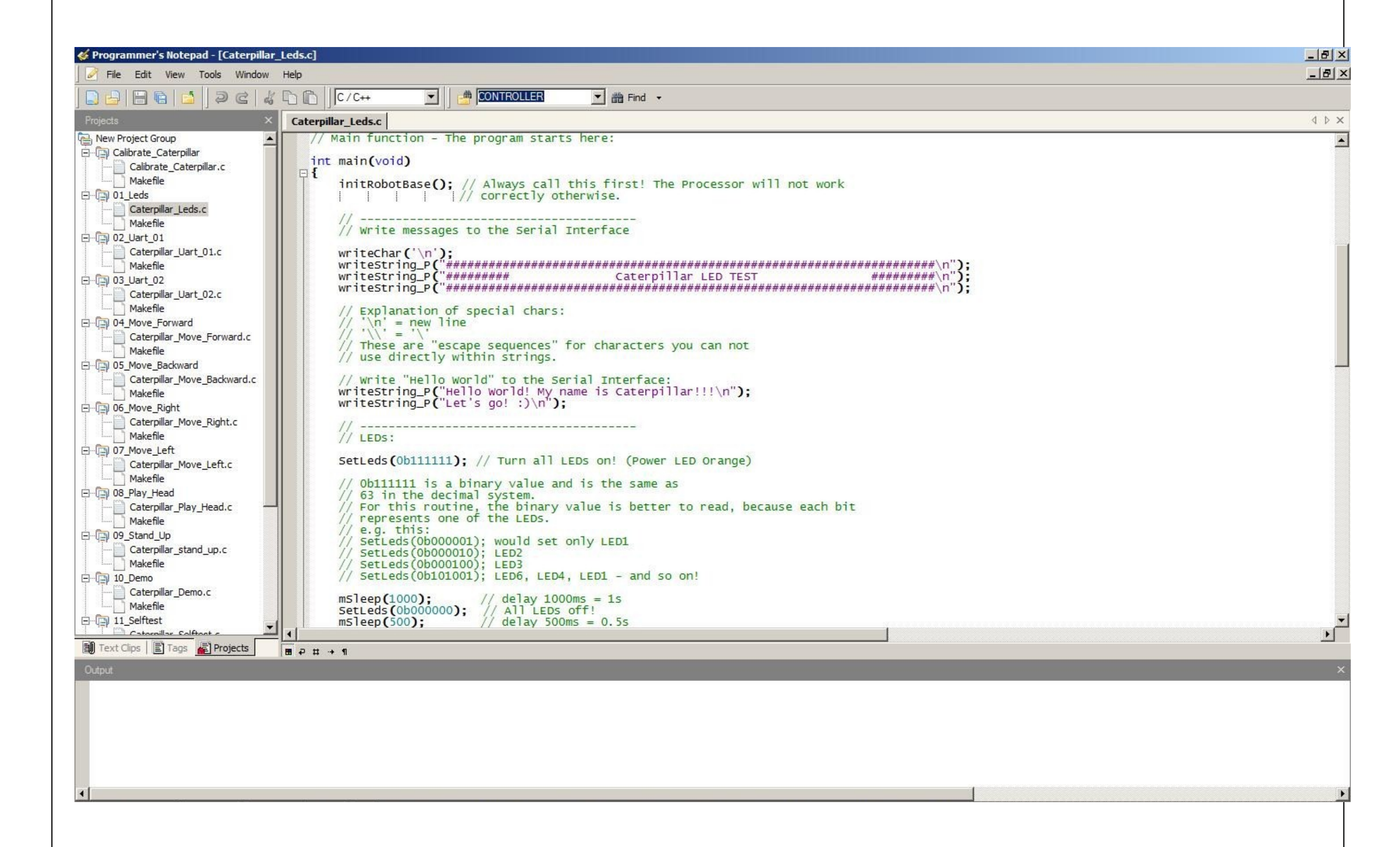
Mit den neuesten Versionen von WINAVR ist es nicht mehr notwendig die Menüeinträge zu erstellen!

ACHTUNG:
In diesem Abschnitt beschreiben wir nicht mehr, wie Sie in PN2 die Menüeinträge erstellen müssen! Mit den neuesten WINAVR Versionen sind diese bereits erstellt.

-

Siehe Seite 37 **„Beispielprojekt öffnen und kompilieren“** wie Sie ein Beispiel Programm öffnen können!

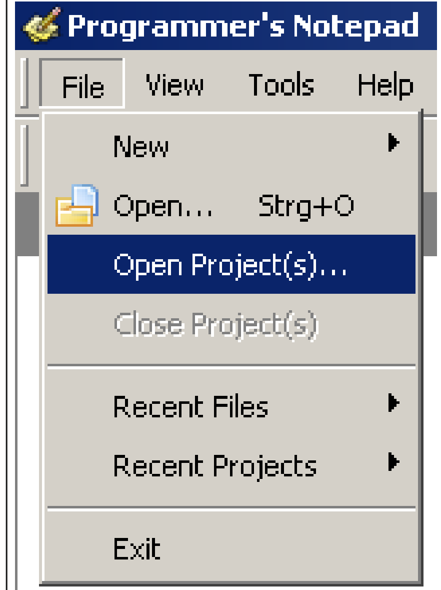
Wenn Sie ein Beispielprojekt geöffnet haben, sollte es im PN2 Schirm *etwa* so aussehen:



Links sind alle Beispielprojekte zu sehen, rechts der Quelltexteditor (mit dem angesprochenen Syntax Highlighting) und unten die Ausgabe der Tools (in diesem Fall die Ausgabe des Compilers).

Sie können noch viele andere Sachen in PN2 umstellen und es gibt viele nützliche Funktionen.

Beispielprojekt öffnen und kompilieren



Jetzt probieren wir gleich mal aus, ob auch alles richtig funktioniert und öffnen die Beispielprojekte:

Im Menü „File" den Menüpunkt „Open Project(s)" wählen.

Es erscheint ein normaler Dateiauswahl Dialog. Hier suchen Sie bitte den Ordner „CaterpillarExamples\" im Ordner in dem Sie die Beispielprogramme gespeichert haben.

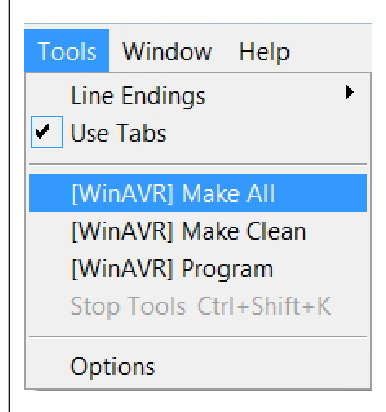
Öffnen Sie nun bitte die Datei „Caterpillar_Examples.ppg". Das ist eine Projektgruppe für PN2, die alle Beispielprogramme sowie die Caterpillar Library in die Projektliste („Projects") lädt. So hat man immer alle Beispielprogramme bequem zur Verfügung und kann sich zu Beginn besser an diesen orientieren oder Funktionen in der Caterpillar Library nachschlagen etc..

Öffnen Sie nun das zweite Beispielprogramm in der Liste („01_Leds" und die Datei „01_Caterpillar_Leds" selektieren), die am linken Rand des Programmfensters zu sehen ist! Dazu einfach doppelt auf „01_Caterpillar_Leds.c" klicken! Es erscheint ein Quelltexteditor in einem Fenster innerhalb des Programms.

Unten im Programmfenster von PN2 sollte ein Ausgabebereich zu sehen sein – falls nicht, müssen Sie diesen Bereich im Menü mit View->Output aktivieren ODER, falls er zu klein ist, durch „ziehen" mit der Maus vergrößern (der Mauscursor verändert sich unten im Programmfenster am oberen Rand des grauen Bereichs in dem „Output" steht in einem recht schmalen Bereich in einen Doppelpfeil...).

Sie können sich das Programm in dem gerade geöffneten Quelltexteditor schonmal kurz anschauen, allerdings müssen Sie hier noch nicht verstehen, was da genau gemacht wird. Das wird weiter unten noch genauer erklärt. Schonmal vorweg: Bei dem grün eingefärbten Text handelt es sich um Kommentare die nicht zum eigentlichen Programm gehören und nur der Beschreibung/Dokumentation dienen. Darauf gehen wird später noch genauer ein. Die Kommentare blähen das schon ziemlich auf, sind aber zur Erklärung notwendig.

Zunächst wollen wir nur ausprobieren, ob das Übersetzen von Programmen korrekt funktioniert.



Es sollten oben im Tools Menü die beiden eben angelegten Menüeinträge (s. Abb.) vorhanden sein (oder die standardmäßig in PN2 vorhandenen [WinAVR] Einträge, das ist egal, es funktioniert normalerweise mit beiden).

Klicken Sie jetzt bitte auf „MAKE ALL"!

Pn2 ruft nun die oben angesprochene „make_all.bat" Batch Datei auf. Diese wiederum ruft das Programm „make" auf. Mehr zu „make" folgt noch später.

Das Beispielprogramm wird nun übersetzt (das nennt man „kompilieren“ vom englischen „to complile“ bzw. „Compiler“=„Übersetzer“) und eine Hexdatei erzeugt. Diese enthält das Programm in der für den Mikrocontroller übersetzten Form und kann dann später in diesen geladen und ausgeführt werden! Es werden während der Kompilierung noch viele temporäre Dateien erzeugt (Endungen wie „.o, .lss, .map, .sym, .elf, .dep“). Die brauchen sie *alle* nicht weiter zu beachten! Mit dem eben angelegten Tool „make clean“ kann man diese bequem löschen. Davon ist nur die Hexdatei für Sie interessant! Die Hexdatei wird beim Aufruf von „make clean“ übrigens nicht gelöscht.

Es sollte nach dem Aufruf des MAKE ALL Menüpunktes folgende Ausgabe erscheinen (hier jedoch stark gekürzt! Einiges kann natürlich etwas anders aussehen):

```
> "make.exe" all

-------- begin --------

avr-gcc (WinAVR 20100110) 4.3.3
Copyright (C) 2008 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Size before:AVR Memory Usage
----------------
Device: atmega16
Program:    2700 bytes (16.5% Full)
(.text + .data + .bootloader)
Data:        67 bytes (6.5% Full)
(.data + .bss + .noinit)

Compiling C: Caterpillar_Leds.c
avr-gcc -c -mmcu=atmega16 -I.
-gdwarf-2 -DF_CPU=16000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wall
-Wstrict-prototypes -Wa,-adhlns=./Caterpillar_Leds.lst  -std=gnu99 -MMD -MP -MF .dep/Caterpillar_Leds.o.d Caterpillar_Leds.c -o
Caterpillar_Leds.o

Linking: Caterpillar_Leds.elf
avr-gcc -mmcu=atmega16 -I. -gdwarf-2 -DF_CPU=16000000UL -Os -funsigned-char -funsigned-bitfields
Creating load file for Flash: Caterpillar_Leds.hex
Creating load file for EEPROM: Caterpillar_Leds.eep
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" \
        --change-section-lma .eeprom=0 --no-change-warnings -O ihex Caterpillar_Leds.elf Caterpillar_Leds.eep || exit 0

Size after:AVR Memory Usage
----------------
Device: atmega16

Program:    3044 bytes (18.6% Full)
(.text + .data + .bootloader)

Data:        63 bytes (6.2% Full)
(.data + .bss + .noinit)

-------- end --------

> Process Exit Code: 0
> Time Taken: 00:04
```

Wichtig ist ganz unten das „**Process Exit Code: 0**“. Das bedeutet, dass es beim Übersetzen keinen Fehler gegeben hat. Steht dort ein anderer Code, gibt es einen Fehler im Quellcode, den man korrigieren muss, bevor es klappt. Der Compiler gibt in diesem Fall weiter oben diverse Fehlermeldungen aus, in denen man mehr Infos dazu findet.

Aber bitte beachten Sie, dass „Process Exit Code: 0“ nicht auf ein komplett fehlerfreies Programm hinweist! Denkfehler in Ihrem Programm findet der Compiler natürlich nicht und er kann auch nicht verhindern, dass der Roboter vor die Wand fährt ;-)

WICHTIG: Weiter oben können auch noch Warnungen u.ä. stehen – diese sind oft sehr sehr hilfreich und weisen fast immer auf wichtige Probleme hin! Daher sollten diese immer beseitigt werden! PN2 hebt Warnungen und Fehler farbig hervor, so dass man diese leicht identifizieren kann. Es wird auch die Zeilennummer angegeben, die der Compiler bemängelt. Wenn man auf diese farbig hervorgehobene Meldung klickt, springt PN2 im entsprechenden Editor direkt zu der jew. Zeile.

Auch sehr hilfreich ist die Angabe zum Schluss „AVR Memory Usage“.

```
Size after:
AVR Memory Usage
----------------
Device: atmega16
Program: 3812 bytes (23.3% Full)
(.text + .data + .bootloader)
Data: 107 bytes (10.7% Full)
(.data + .bss + .noinit)
```

Das bedeutet hier, beim Atmega16 Prozessor, dass unser Programm 3812 Bytes groß ist und 107 Bytes RAM für statische Variablen reserviert sind (dazu kommen noch die dynamischen Bereiche für Heap und Stack, das würde an dieser Stelle aber zu weit führen... halten Sie einfach immer mindestens ein paar hundert Bytes Speicher frei). Wir haben insgesamt 16KB (16384 Bytes) an Flash ROM und 1KB (1024 Bytes) an RAM. Von den 16KB sind 2K mit dem Bootloader belegt – also können wir nur 14KB nutzen. Immer darauf achten, dass das Programm auch noch in den verfügbaren Speicher passt! (Der RobotLoader überträgt das Programm nicht wenn es zu groß ist!)

Bei dem Beispielprogramm oben sind also noch 10416 Bytes frei. Das eigentlich recht kurze Beispielprogramm Example_01_Caterpillar_Leds.c ist übrigens nur deshalb schon so groß, weil die CaterpillarBaseLibrary mit eingebunden wird! Also keine Sorge, es ist genug Platz für Ihre Programme vorhanden und so kleine Programme brauchen normalerweise nicht so viel Speicher. Die Funktionsbibliothek benötigt alleine nämlich schon mehrere KB vom Flashspeicher, nimmt Ihnen aber auch sehr viel Arbeit ab und daher werden Ihre eigenen Programme meist relativ klein sein im Vergleich zur CaterpillarBaseLibrary.

Das eben kompilierte Programm kann nun mit dem RobotLoader in den Roboter geladen werden. Dazu fügen Sie die eben erzeugte Hexdatei in die Liste im RobotLoader mit „Add“ bzw. „Hinzufügen“ ein, selektieren diese und klicken auf den „Upload!“ Button, genau wie Sie es auch schon beim Selbsttestprogramm getan haben. Danach können Sie wieder auf den Terminal wechseln und sich die Ausgabe des Programms anschauen. Die Programmausführung muss dazu natürlich zunächst wieder gestartet werden, im Terminal ist es am bequemsten [STRG]+[S] auf der Tastatur zu drücken oder das Menü zu benutzen (oder einfach ein „s“ senden – nach einem Reset müssen Sie allerdings immer etwas warten, bis die Meldung „[READY]“ im Terminal erscheint!). Auch [STRG]+[Y] ist eine sehr nützliche Tastenkombination, denn damit wird das aktuell selektierte Programm in den Caterpillar geladen und direkt danach gestartet! Man muss also nicht extra vom Terminal aus wieder auf den Karteireiter „Flash Loader“ wechseln oder das Menü benutzen.

Das Beispielprogramm ist sehr einfach und besteht nur aus einem kleinen LED Lauflicht und etwas Textausgabe.

Bevor Sie nun Ihre eigenen Programme schreiben können, folgt ein kleiner C Crashkurs...

-

6.1. Warum ausgerechnet C? Und was bedeutet „GCC“?

Die Programmiersprache C ist sehr weit verbreitet – es ist die Standardsprache, die eigentlich jeder, der sich für Softwareentwicklung interessiert, mal verwendet haben sollte (oder zumindest von der Syntax her ähnliche Sprachen). Für so gut wie jeden derzeit verfügbaren Mikrocontroller existiert mindestens ein C Compiler. Aus diesem Grund können alle neueren Roboter von AREXX Engineering (zur Zeit ASURO, YETI und der RP6) in C programmiert werden.

Da C sehr weit verbreitet ist, gibt es sehr viel Dokumentation dazu im Internet und in Büchern. Das macht es Einsteigern natürlich einfacher, auch wenn C schon eine relativ komplexe Sprache ist, die man ohne Vorkenntnisse normalerweise *nicht mal eben so* innerhalb von zwei drei Tagen erlernen kann... (also bitte nicht gleich den Roboter aus dem Fenster werfen, wenn es mal nicht auf Anhieb klappen sollte ;-))

Die Grundlagen sind zum Glück einfach zu verstehen und man kann seine Fähigkeiten kontinuierlich ausbauen und verbessern. Das erfordert aber schon etwas Eigeninitiative! Von alleine lernt sich C nicht – das ist ähnlich wie mit normalen Fremdsprachen! Wenn man C aber erstmal einigermaßen beherrscht, ist der Einstieg in viele andere Programmiersprachen auch kein allzugroßes Problem mehr, da oft sehr ähnliche Konzepte verwendet werden.

Für den Caterpillar kommt wie auch bei unseren anderen Robotern eine spezielle Version des C Compilers aus der GNU Compiler Collection oder kurz GCC zum Einsatz. Es handelt sich beim GCC um ein universelles Compiler System, welches verschiedenste Sprachen unterstützt. So kann man damit z.B. neben C auch in C++, Java, Ada und FORTRAN verfasste Quelltexte übersetzen.

Der GCC unterstützt nicht nur den AVR, sondern wurde eigentlich für viel größere Systeme entwickelt und kennt einige dutzend verschiedene Zielsysteme.

Prominentestes Projekt, für das der GCC verwendet wird, ist natürlich Linux. Auch fast alle Anwendungsprogramme die unter Linux laufen, wurden ebenfalls mit dem GCC übersetzt. Es handelt sich hier also um ein sehr ausgereiftes und professionelles Werkzeug, das auch in vielen großen Firmen zum Einsatz kommt.

Übrigens: Wenn wir von „GCC“ sprechen, meinen wir in diesem Handbuch nicht unbedingt die komplette Compiler Collection, sondern fast immer nur den C-Compiler. Ursprünglich stand GCC sogar nur für „GNU C Compiler“ - die neuere Bedeutung wurde notwendig, als auch andere Sprachen hinzukamen.

Wenn Sie mehr über den GCC erfahren möchten, können Sie die offizielle Homepage besuchen:
http://gcc.gnu.org/

Der GCC unterstützt den AVR nicht sofort von sich aus und muss erst angepasst werden. Diese Version des GCC nennt sich dann AVR-GCC. Dieser Compiler ist für Windows Benutzer fertig eingerichtet in WinAVR enthalten. Bei Linux muss man sich diesen meist noch selbst übersetzen, was Sie ja hoffentlich schon erledigt haben.

6.2. C - Crashkurs für Einsteiger

ACHTUNG!

Dieser C-Crashkurs wurde teilweise aus der RP6 Anleitung übernommen und an einigen Stellen in dieser Caterpillar Anleitung tauchen daher Beispiele des RP6 auf, da diese bereits sehr ausführlich beschrieben sind und die Grundlagen für das C Programmieren von RP6 und Caterpillar natürlich gleich sind.

Auf der Caterpillar CD finden Sie natürlich die Caterpillar Library und Caterpillar Beispielprogramme.

Dieses Kapitel gibt Ihnen ***nur eine ganz kurze Einführung*** *in die C Programmierung. Wir besprechen hier nur die absolut notwendigen Dinge, die man für den Caterpillar unbedingt braucht. Dieser Abschnitt und viele der Beispielprogramme verstehen sich eher als Kurzübersicht: „Was es so alles gibt und was man alles machen kann". Es werden Beispiele gegeben und die Grundlagen erklärt, aber es wird dem Leser überlassen, sich dann ausführlicher damit zu befassen!*
Es ist also nicht mehr als ein kleiner Crashkurs! Eine vollständige Einführung würde den Rahmen dieser *Bedienungsanleitung* bei weitem sprengen und füllt normalerweise dicke Fachbücher! Davon gibt es aber glücklicherweise sehr viele! Einige[1] davon sind sogar kostenlos im Internet verfügbar - hier folgt nur eine kleine Übersicht...

Literatur

Die folgenden Bücher und Tutorials beschäftigen sich mit der C Programmierung hauptsächlich für den PC und andere "große" Rechner. *Vieles, was in diesen Büchern steht, gibt es nicht für AVR Mikrocontroller - die Sprache ist zwar gleich, aber die meisten Bibliotheken, die man auf normalen PCs verwenden kann, sind für einen Mikrocontroller schlicht zu groß. Bestes Beispiel sind Funktionen wie „printf" - auf dem PC eine Selbstverständlichkeit! Diese Funktion gibt es zwar auch für Mikrocontroller, jedoch braucht sie ziemlich viel Speicher und Rechenzeit und sollte daher besser nicht verwendet werden. Wir besprechen später noch genügend für unsere Zwecke effektivere Alternativen.*

Einige C Tutorials / Online-Bücher (nur eine winzig kleine Auswahl):

http://www.galileocomputing.de/openbook/c_von_a_bis_z/
http://de.wikibooks.org/wiki/C-Programmierung
http://suparum.rz.uni-mannheim.de/manuals/c/cde.htm
http://www.roboternetz.de/wissen/index.php/C-Tutorial
http://www.its.strath.ac.uk/courses/c/

Weiterhin gibt es viele Bücher auf dem Markt - einfach mal in eine gut sortierte Bibliothek gehen oder bei Buchhändlern stöbern! Da findet sich meist eine ganze Menge.

Sie müssen sich kein Buch kaufen, wenn Sie nur ein wenig mit dem Roboter herumexperimentieren wollen - vieles muss man sich ohnehin durch "Learning by doing" beibringen! Alle benötigten Informationen finden sich auch auf den genannten Seiten im Internet und die auf der CD mitgelieferten Beispielprogramme sind auch schon recht umfangreich.

Speziell für den AVR-GCC und AVR Mikrocontroller gibt es ein sehr gutes deutschsprachiges Tutorial, und zwar hier:

http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial

Dort werden einige Dinge genannt (Programmiergeräte etc.), die man für den Caterpillar aber nicht braucht. Das meiste davon ist jedoch sehr nützlich und es lohnt sich auf jeden Fall, dort mal reinzuschauen!

Ein weiterer Artikel, der sich aber eher mit dem Compiler an sich und einigen speziellen Bereichen befasst, ist hier zu finden:

http://www.roboternetz.de/wissen/index.php/Avr-gcc

Dann gibt es auch noch die WinAVR Homepage und Dokumentation:

http://winavr.sourceforge.net/
http://winavr.sourceforge.net/install_config_WinAVR.pdf
sowie die AVR-LibC Dokumentation:
http://www.nongnu.org/avr-libc/user-manual/index.html

Sie müssen natürlich nicht alle diese Tutorials/Bücher lesen! Diese Liste hier versteht sich zur Auswahl! Aber nicht alle Tutorials sind gleichermaßen ausführlich und viele behandeln sehr unterschiedliche Themen. Es lohnt sich also schon, mehrere verschiedene zu lesen.

Erstes Beispielprogramm

Wie gesagt - learning by doing ist der beste Weg, um die Sprache C zu lernen. Wenn Sie also etwas in diesem Crashkurs gelesen und soweit verstanden haben, sollten Sie es auch ausprobieren! Probieren geht hier wirklich über Studieren!

Natürlich müssen noch ein paar Gundlagen besprochen werden, aber damit Sie direkt einen Eindruck davon bekommen, worüber wir hier eigentlich die ganze Zeit reden, schauen wir uns jetzt mal an, wie ein kleines C Programm für den Caterpillar typischerweise ausschaut:

```
1	/*
2	* Ein kleines "Hallo Welt" C Programm für den Caterpillar!
3	*/
4
5	#include "CaterpillarBaseLib.h"
6	int main(void)
7
8	{
9		initRobotBase();
10		writeString("Hallo Welt!\n");
11		return 0;
12	}
```

Wenn Sie noch nie zuvor in C programmiert haben, sieht das wahrscheinlich auf den ersten Blick aus wie eine Fremdsprache (und das ist es ja auch!), aber die Grundlagen sind wirklich sehr einfach zu verstehen. C wurde im englischsprachigen Raum entwickelt*[2] und deshalb sind auch alle Befehle an die englische Sprache angelehnt. Das ist aber nicht nur bei C so, sondern bei so gut wie allen Programmiersprachen.

2 ...genauer gesagt zu Beginn der 1970er Jahre in den USA, wo es dann als Entwicklungsgrundlage des UNIX Betriebssystems gedient hat – es folgten später noch viele Verbesserungen und Ergänzungen...

Wir halten uns bei den größeren Beispielprogrammen auch konsequent daran und benennen alle Funktionen und Variablen in den Beispielprogrammen mit englischen Begriffen! Hier im Handbuch und diesem C Crashkurs wird jedoch ab und an auch mal eine Ausnahme gemacht.

Bei den Programmen dieses Crashkurses sind die Kommentare auf Deutsch verfasst, aber bei den richtigen Beispielprogrammen und der CaterpillarLibrary sind sie komplett auf Englisch.

Das hat den einfachen Grund, dass man ansonsten zwei Versionen des Quellcodes pflegen müsste und jedes Mal, wenn etwas geändert wird oder neue Funktionen hinzukommen, alles in zweifacher Ausführung ändern und in zwei Sprachen kommentieren müsste. Das macht nicht nur viel Arbeit, sondern verzögert auch die Freigabe von Verbesserungen und neuen Versionen.

Wer C lernen will bzw. sich detaillierter mit Mikrocontrollern, Elektronik und Robotik befasst, kommt ohnehin nur schwer um Englisch herum.

Allerdings wird die Funktionsweise der Beispielprogramme auch noch im entsprechenden Kapitel auf Deutsch erläutert!

Nun aber zurück zum Beispiel. Dieses kleine Programm oben ist zwar funktionsfähig, tut aber nichts weiter als den Mikrocontroller zu initialisieren und den Text:

"Hallo Welt!" + Zeilenvorschub / Neue Zeile

über die serielle Schnittstelle auszugeben! Ein typisches Programm, das in so gut wie jedem Buch zu finden ist. Das kleine Beispielprogramm können Sie natürlich auch ausprobieren. Es kann sehr sinnvoll sein, das einfach mal abzutippen, um ein Gefühl für die Sprache zu bekommen! Vor allem an die vielen Semikolons und Sonderzeichen muss man sich erstmal gewöhnen...

Wem das obige Programm aber schon zu langweilig ist: Auf der CD findet sich bei den Beispielprogrammen noch ein etwas interessanteres "Hallo Welt" Programm mit kleinem LED-Lauflicht

Wir werden nun das Programm in Listing 1, Zeile für Zeile durchgehen und erklären!

Zeile 1 - 3: `/* Ein kleines "Hallo Welt" C Programm für den Caterpillar! */`

Das ist ein Kommentar. Dieser Teil des Quelltextes wird vom Compiler nicht weiter beachtet. Kommentare werden zur Dokumentation des Quelltextes verwendet und erleichtern das Verständnis fremder (und eigener!) Quelltexte. Bei fremden Quelltexten kann man damit besser nachvollziehen, was sich der Programmierer dabei gedacht hat und bei eigenen Quelltexten hat man auch nach ein paar Jahren noch gute Anhaltspunkte, um sich an die eigenen Gedankengänge besser erinnern zu können. Kommentare beginnen mit /* und enden mit */

Sie können dazwischen beliebig lange Kommentare verfassen, oder auch Teile Ihres Quellcodes „auskommentieren“, um z.B. eine andere Variante zu testen, ohne den alten Quellcode zu löschen. Neben diesem mehrzeilen Kommentaren unterstützt der GCC auch einzeilige Kommentare, die mit einem „//“ eingeleitet werden. Alles in einer Zeile nach einem „//“ wird vom Compiler als Kommentar interpretiert.

Zeile 5: #include "CaterpillarBaseLib.h“

Hier binden wir die Caterpillar Funktionsbibliothek ein. Diese stellt sehr viele nützliche Funktionen und vordefinierte Dinge bereit, die das Ansteuern der Hardware immens erleichtern. Das müssen wir über eine sog. Headerdatei (Endung „*.h“) einbinden, da der Compiler sonst nicht weiss, wo er die ganzen Funktionen finden kann. Header braucht man für alle Dinge, die in externen C-Dateien liegen. Wenn Sie sich mal den Inhalt von der CaterpillarBaseLib.h und dann noch der CaterpillarBaseLib.c ansehen, werden Sie das Prinzip wahrscheinlich besser verstehen. Mehr zu „#include“ folgt im Kapitel über den Präprozessor.

Zeile 7: int main(**void**)

Das Wichtigste an dem Beispielprogramm: Die Main Funktion. Was eine Funktion genau ist, wird etwas später erklärt. Zunächst reicht es uns zu wissen, dass das Programm hier anfängt (engl. „Main Function“ bedeutet zu deutsch „Haupt-Funktion“).

Zeile 8 und Zeile 12: { }

In C werden sogenannte "Blöcke" mit geschweiften Klammern - also '{' und '}' definiert! (Auf der Tastatur sind das [AltGr] + [7] für '{' und [AltGr] + [0] für '}'). Blöcke fassen mehrere Befehle zusammen.

Zeile 9: initRobotBase**();**

Hier wird eine Funktion aus der Caterpillar library aufgerufen, die den AVR Mikrocontroller initialisiert. Damit werden die Hardware Funktionen des AVRs konfiguriert. Ohne diesen Aufruf arbeiten die meisten Funktionen des Mikrocontrollers nicht korrekt! Also niemals vergessen, das als erstes aufzurufen!

Zeile 10: writeString**(**"Hello World!\n"**);**

Hier wird die Funktion "writeString" aus der Caterpillar Library mit dem Text "Hello World!\n" als Parameter aufgerufen. Der Text wird dann von dieser Funktion über die serielle Schnittstelle ausgegeben.

Zeile 11: return 0;

Hier endet das Programm – die Main Funktion wird verlassen und der Wert 0 zurückgegeben.Das wird auf großen Computern meist für Fehlercodes oder ähnliches benutzt. Auf dem Mikrocontroller braucht man das aber eigentlich gar nicht und es ist nur da, weil es der C Standard so will.

Jetzt haben Sie schonmal einen kleinen Eindruck davon bekommen, wie so ein C-Programm ausschaut. Nun müssen wir aber erstmal noch ein paar Grundlagen besprechen, bevor es weitergeht.

-

C Grundlagen

Wie schon zu Beginn gesagt, besteht ein C Programm aus reinem ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchage) Text. Es wird dabei strikt zwischen Groß- und Kleinschreibung unterschieden. Das heißt, Sie müssen eine Funktion, die "ichBinEineFunktion" heißt, auch immer genau so aufrufen! Der Aufruf: "IcHbInEiNeFuNkTIoN();" würde so nämlich nicht funktionieren!

Man kann zwischen den einzelnen Befehlen und Anweisungen beliebig viele Leerzeichen, Tabulatoren und Zeilenumbrüche einfügen, ohne das sich die Bedeutung des Programms ändert.

Sie haben ja schon bei dem Beispielprogramm gesehen, dass die Befehle mit Tabulatoren eingerückt worden sind, um das Programm besser lesbar zu machen. Notwendig ist das aber nicht! Den Teil ab Zeile 7 vom Listing oben könnte man z.B. auch so schreiben:

```
1 int main(void){initRobotBase();writeString("Hallo Welt!\n");return 0;}
```

Das ist von der Bedeutung her identisch – aber etwas unübersichtlich. Hier wurden nur die Zeilenumbrüche und Einrückungen entfernt! Dem Compiler ist das völlig egal! (Einige Leerzeichen z.B. zwischen „int" und „main" sind natürlich immer notwendig um einzelne Schlüsselwörter und Bezeichner zu trennen und innerhalb von Anführungszeichen darf man auch nicht einfach so einen Zeilenumbruch einfügen!)

Mehrere Anweisungen werden mit den oben angesprochenen geschweiften Klammern { } zu sog. Blöcken zusammengefasst. Das brauchen wir bei Funktionen, Bedingungen und Schleifen.

Jede Anweisung wird mit einem Semikolon ';' beendet, so kann der Compiler die Anweisungen auseinanderhalten.

Ein wichtiger Hinweis gleich zu Beginn, wenn Sie die Programme in diesem Tutorial abtippen wollen: Man vergisst sehr leicht, die Befehle jeweils mit einem Semikolon zu beenden – oder setzt eins an eine falsche Stelle und wundert sich dann, warum das Programm nicht das tut, was es soll! Vergisst man ein einziges Semikolon in bestimmten Programmteilen, kann der Compiler übrigens gleich einen ganzen Haufen von Fehlermeldungen erzeugen – auch wenn es eigentlich nur ein einziger Fehler ist. Meist zeigt schon die erste Fehlermeldung den tatsächlichen Fehler an.

Auch vergisst man gern, eine der vielen Klammern zu schließen oder vertut sich mit der Schreibweise von Befehlen. Der Compiler verzeiht keine Syntax Fehler (= „Rechtschreibfehler")! Das ist vielleicht erstmal gewöhnungsbedürftig, sich an all diese Regeln zu halten, aber das lernt man nachdem man einige Programme selbst geschriebenhat relativ schnell.

Jedes C Programm beginnt in der Main Funktion. Die dann folgenden Befehle werden grundsätzlich sequentiell, Anweisung nach Anweisung von oben nach unten abgearbeitet. Mehrere Befehle gleichzeitig kann ein AVR Mikrocontroller nicht ausführen!

Das macht aber nichts, denn es gibt viele Möglichkeiten, den Progammablauf zu beeinflussen und an andere Stellen im Programm zu springen (mit sog. Flusssteuerung, dazu kommen wir später).

Variablen

Erstmal fangen wir aber damit an, wie wir denn überhaupt Daten im Arbeitsspeicher ablegen und auslesen können. Das geht mit Variablen. Es gibt in C verschiedene Arten von Variablen - sog. Datentypen. Grundsätzlich sind dies 8, 16 oder 32Bit große, ganzzahlige Datentypen, die entweder mit oder ohne Vorzeichen verwendet werden können. Wieviel "Bit" eine Variable haben muss, hängt davon ab wie groß die Zahlen sind (Wertebereich), die darin gespeichert werden sollen.

Beim Caterpillar verwenden wir die folgenden Datentypen:

Typ	Alternative	Wertebereich	Bemerkung
signed char	**int8_t**	8 Bit: **-128** ... **+127**	1 Byte
char	**uint8_t**	8 Bit: **0** ... **255**	' ' vorzeichenlos
int	**int16_t**	16 Bit: **-32768** ... **+32767**	2 Bytes
unsigned	**int uint16_t**	16 Bit: **0** ... **65535**	" vorzeichenlos
long	**int32_t** 32 Bit:	**–2147483648** ... **+2147483647**	4 Bytes
unsinged long	**uint32_t** 32 Bit:	**0** ... **4294967295**	" vorzeichenlos

Da insbesondere der Datentyp „int“ auf verschiedenen Plattformen nicht standardisiert ist und z.B. auf unserem Mikrocontroller 16 bit, auf PCs aber 32 bit groß ist, verwendenwir die neuere standardisierte Bezeichnung: int16_t

Diese Datentypen ist immer so aufgebaut: [u] int N _t
u : unsigned (kein Vorzeichen)
int : Integer (ganze Zahl)
N : Anzahl der Bits, z.B. 8, 16, 32 oder 64
_t : t wie „type“ um Verwechslungen mit anderen Bezeichnern auszuschließen.

Auf einem Mikrocontroller kommt es oft auf jedes Byte Speicherplatz an, daher behält man mit dieser Bezeichnung auch besser den Überblick. Man sieht der Bezeichnung sofort an, dass es sich um einen 16bit Datentyp handeln muss (wegen der 16 im Namen). Steht noch ein „u“ davor, handelt es sich um einen „unsigned“ also vorzeichenlosen Datentyp. Sonst ist der Datentyp „signed“ also vorzeichenbehaftet.

Das wurde oben in der Tabelle aber nur bei „signed char“ hingeschrieben, denn int und long sind standardmäßig vorzeichenbehaftet und nur char ist standardmäßig vorzeichenlos - auch wenn man es nicht explizit hinschreibt. Das resultiert aus einer Compileroption, die wir verwenden, und die auch fast immer beim AVR-GCC aktiviert ist.

Bei Strings (=Engl. für Zeichenketten) wird auch weiterhin „char“ genutzt, da sonst wegen der Definition von uint8_t ein paar Dinge aus der C Standard Bibliothek nicht damit kompatibel wären und es ohnehin logischer ist, hierfür den Datentyp char (=Engl. für „Zeichen“) zu verwenden. Das wird im Kapitel über die Caterpillar Library bei Textausgabe über die serielle Schnittstelle noch etwas genauer behandelt. Also merken wir uns einfach: Bei Zeichen und Zeichenketten immer „char“ verwenden, bei Zahlenwerten immer uintN_t oder intN_t!

Um eine Variable nutzen zu können, muss sie zunächst deklariert werden. Hierbei wird festgelegt, welchen Datentyp, Namen und evtl. noch welchen Anfangswert die Variable erhalten soll. Der Name einer Variablen muss dabei mit einem Buchstaben beginnen (Der Unterstrich also "_" zählt ebenfalls als Buchstabe) und darf auch Zahlen, aber *keine* Sonderzeichen wie z.B. „äöüß#'[]²³|*+-.,<>%&/(){}$§=´°?!^“ o.ä. enthalten.

Groß- und Kleinschreibung werden wie immer unterschieden. Also sind aBc und abC verschiedene Variablen! Traditionellerweise werden immer Kleinbuchstaben zumindest für den Anfangsbuchstaben von Variablennamen verwendet. Folgende Bezeichnungen sind bereits für andere Dinge reserviert und können NICHT als Variablennamen, Funktionsnamen oder jegliche andere Bezeichner verwendet werden:

```
auto      default  float  long      sizeof   union
break     do       for    register  static   unsigned
case      double   goto   return    struct   void
char      else     if     short     switch   volatile
const     enum     int    signed    typedef  while
continue  extern
```

Es gibt weiterhin noch Fließkommazahlen der Typen float und double. Man sollte auf einem kleinen AVR Mikrocontroller aber möglichst vermeiden, damit zu arbeiten. Kostet meistens zuviel Rechenzeit und Speicherplatz und man kommt oft mit normalen ganzen Zahlen besser aus. Für den Caterpillar brauchen wir diese Datentypen also normalerweise nicht.

Eine Variable zu deklarieren ist sehr einfach, hier z.B. ein char mit dem Namen x:

```
char x;
```

Nach dieser Deklaration ist die Variable x im danach folgenden Programmcode gültig und kann verwendet werden.

Man kann ihr z.B. mit

```
x = 10;
```

den Wert 10 zuweisen. Man kann der Variablen auch gleich bei der Deklaration einen Wert zuweisen:

```
char y = 53;
```

Die normalen Grundrechenarten können ganz normal verwendet werden:

```
signed char z; // Char mit Vorzeichen!
z = x + y; // z hat nun den Wert z = x + y = 10 + 53 = 63
z = x – y; // z hat nun den Wert z = 10 – 53 = -43
z = 10 + 1 + 2 – 5; // z = 8
z = 2 * x; // z = 2 * 10 = 20
z = x / 2; // z = 10 / 2 = 5
```

Es gibt dann noch ein paar nützliche Abkürzungen:

```
z += 10; // entspricht: z = z + 10; also ist z = 15
z *= 2; // z = z * 2 = 30
z -= 6; // z = z - 6 = 24
z /= 4; // z = z / 4 = 8
z++; // Abkürzung für z = z + 1; Also ist z hier 9
z++; // z = 10 // z++ nennt man auch „z inkrementieren"
z++; // z = 11 ...
z--; // z = 10 // z-- nennt man auch „z dekrementieren"
z--; // z = 9
z--; // z = 8 ...
```

Oben haben wir noch den Datentyp „char" verwendet. Wie aber schon gesagt, werden in allen anderen Programmen des Caterpillar (fast) nur standardisierte Datentypen verwendet.

So z.B. wäre folgendes: `int8_t x;`

identisch zu: `signed char x;`

Und: `uint8_t x;`

identisch zu: `unsigned char x;` // bzw. bei uns eigentlich auch einfach nur „char", da dieser Datentyp standardmäßig „unsigned" ist.

Bedingungen

Bedingungen mit „if-else" Konstrukten sind sehr, sehr wichtig, um den Programmablauf zu steuern. Man kann mit ihnen überprüfen, ob eine gegebene Bedingung wahr oder falsch ist und einen je nachdem bestimmten Programmcode ausführen oder nicht.

Direkt ein kleines Beispiel:

```
1     uint8_t x = 10;
2     if(x == 10)
3     {
4     writeString("x ist gleich 10!\n");
5     }
```

Hier wird in Zeile 1 zunächst die 8-Bit Variable x deklariert und ihr der Wert 10 zugewiesen. Jetzt überprüfen wir in der nachfolgenden if-Bedingung in Zeile 2, ob die Variable x den Wert 10 hat. Das ist hier natürlich immer der Fall und somit wird der Block unter der Bedingung ausgeführt und „x ist gleich 10!" ausgegeben. Hätten wir x stattdessen mit 231 initialisiert, wäre an dieser Stelle nichts ausgegeben worden!

Allgemein hat eine if-Bedingung immer folgende Syntax:

```
if ( <Bedingung X> )
<Anweisungsblock Y>
else
<Anweisungsblock Z>
```

Ins Deutsche übersetzt heisst das übrigens soviel wie: „wenn X dann Y sonst Z".

Ein weiteres Beispiel dazu:

```
1     uint16_t tolleVariable = 16447;
2     if(tolleVariable < 16000) // Wenn tolleVariable < 16000
3     { // Dann:
4     writeString("tolleVariable ist kleiner als 16000!\n");
5
6     }
7     else // Sonst:
8     {
9     writeString("tolleVariable ist größer oder gleich 16000!\n");
10    }
```

Hier wäre die Ausgabe „tolleVariable ist größer oder gleich 16000!", denn tolleVariable ist hier 16447 und somit größer als 16000. Die Bedingung ist also nicht erfüllt und der else Teil wird ausgeführt. Wie man am Namen „tolleVariable" erkennen kann, hat man bei der Namensgebung von Variablen (und allem anderen) keine anderen Einschränkungen als die weiter oben genannten.

Man kann auch mehrere If-then-else-Bedingungen hintereinander verwenden, um mehrere alternative Fälle abzufragen:

```
1     if(x == 1) { writeString("x ist 1!\n"); }
2     else if(x == 5) { writeString("x ist 5!\n"); }
3     else if(x == 244) { writeString("x ist 244!\n"); }
4     else { writeString("x ist hat einen anderen Wert!\n");}
```

Innerhalb der Bedingungen kann man folgende Vergleichsoperatoren verwenden:

x == y logischer Vergleich auf Gleichheit
x != y logischer Vergleich auf Ungleichheit
x < y logischer Vergleich auf „kleiner"
x <= y logischer Vergleich auf „kleiner oder gleich"
x > y logischer Vergleich auf „größer"
x >= y logischer Vergleich auf „größer oder gleich"

Dann gibt es noch logische Verknüpfungsoperatoren:

x && y wahr, wenn x wahr und y wahr sind
x || y wahr, wenn x wahr und/oder y wahr
!x wahr, wenn x nicht wahr ist

Das kann man alles beliebig miteinander verknüpfen und verschachteln und beliebig viele Klammern setzen:

```
1     if( ((x != 0) && (!(x > 10))) || (y >= 200)) {
2     writeString("OK!\n");
3     }
```

Die obige if Bedingung wird wahr, wenn x ungleich 0 (x != 0) UND x nicht größer als 10 ist (!(x > 10)) ODER wenn y größer oder gleich 200 ist (y >= 200). Dort könnte man noch beliebig viele weitere Bedingungen hinzufügen, sofern notwendig.

Switch-Case

Oft muss man eine Variable auf viele verschiedene Zahlenwerte überprüfen und abhängig davon Programmcodes ausführen. Das kann man natürlich wie oben beschrieben mit vielen if-then-else Bedingungen machen, aber es geht auch eleganter mit einer switch-Verzweigung.

Ein Beispiel:

```
1    uint8_t x = 3;
2
3    switch(x)
4    {
5    case 1: writeString("x=1\n"); break;
6    case 2: writeString("x=2\n"); break;
7    case 3: writeString("x=3\n"); // Hier fehlt das "break", also fährt
8    case 4: writeString("Hallo\n"); // das Programm direkt mit dem
9    case 5: writeString("du\n"); // nächsten und dem nächsten Fall fort,
10   case 6: writeString("da!\n"); break; // und stoppt erst hier!
11   case 44: writeString("x=44\n"); break;
12   // Hier gelangt das Programm hin, wenn keiner der obigen
13   // Fälle gepasst hat:
14   default : writeString("x ist was anderes!\n"); break;
15   }
```

Das ist vom Resultat her *ähnlich* zu dem Beispiel auf der vorherigen Seite mit „if-elseif-else-if-else...“, nur eine andere Schreibweise.

Die Ausgabe wäre in diesem Fall (mit x = 3):

```
x=3
Hallo
du
da!
```

Wenn man x = 1 setzen würde, wäre die Ausgabe „x=1\n“ und mit x=5 wäre die Ausgabe:

```
du
da!
```

Hier sieht man, dass das „break“ die Switch-Verzweigung beendet. Lässt man es weg, läuft das Programm einfach solange durch alle anderen Fälle durch, bis entweder das Ende der switch-Verzweigung erreicht ist oder ein anderes „break“ ausgeführt wird. Dabei ist es egal ob die nachfolgenden Bedingungen erfüllt werden oder nicht!

Wenn x = 7 gesetzt wird, würde keiner der Fälle diesen Wert abdecken, das Programm im default Teil landen und die Ausgabe wäre: „x ist was anderes!\n“.

Diese ganzen Textausgaben sind natürlich nur Beispiele – man könnte in realen Programmen für den Roboter damit z.B. verschiedene Bewegungsmuster auslösen. In einigen Anwendungsbeispielen werden switch-case Konstrukte z.B. für endliche Automaten verwendet, um verschiedene Verhalten des Roboters zu implementieren.

Schleifen

Schleifen werden immer dann gebraucht, wenn bestimmte Operationen mehrmals wiederholt werden müssen.

Dazu direkt ein kleines Beispiel:

```
1     uint8_t i = 0;
2     while(i < 10) // solange i kleiner als 10 ist...
3     { // ... wiederhole folgenden Programmcode:
4     writeString("i="); // "i=" ausgeben,
5     writeInteger(i, DEC); // den dezimalen (engl. "DECimal") Zahlenwert
6     // von i und ...
7     writeChar('\n'); // ... einen Zeilenumbruch ausgeben.
8     i++; // i inkrementieren.
9     }
```

Hier handelt es sich um eine „while“ Schleife, die der Reihe nach „i=0\n“, „i=1\n“, „i=2\n“, ... „i=9\n“ ausgibt. Der Block nach dem sog. Schleifenkopf, also dem „while(i < 10)“ wird solange wiederholt, wie die Bedingung wahr ist. In diesem Fall ist die Bedingung „i ist kleiner als 10“. Auf Deutsch würde das obige also soviel wie „wiederhole den folgenden Block, solange i kleiner als 10 ist“ heissen. Da i hier zunächst 0 ist und bei jedem Schleifendurchlauf um 1 erhöht wird, läuft die Schleife insgesamt 10 mal durch und gibt die Zahlen von 0 bis 9 aus. Die Bedingung im Schleifenkopf kann genau so aufgebaut werden, wie bei den if Bedingungen.

Neben der while Schleife gibt es noch die „for“ Schleife. Die funktioniert ganz ähnlich, nur kann man hier noch etwas mehr im Schleifenkopf unterbringen.

Dazu wieder ein Beispiel:

```
1     uint8_t i; // wird hier NICHT initialisiert, sondern im Schleifenkopf!
2     for(i = 0; i < 10; i++)
3     {
4     writeString("i=");
5     writeInteger(i, DEC);
6     writeChar('\n');
7     }
```

Diese Schleife erzeugt exakt dieselbe Ausgabe wie die obige while Schleife. Hier ist nur noch einiges mehr im Schleifenkopf untergebracht.

Der grundsätzliche Aufbau ist folgender:

```
for ( <Laufvariable initialisieren> ; <Abbruchbedingung> ; <Laufvariable verändern> )
{
      <Anweisungsblock>
}
```

Für Mikrocontroller braucht man oft Endlosschleifen, also Schleifen die „unendlich“ oft wiederholt werden. Fast jedes Mikrocontrollerprogramm hat eine Endlosschleife – sei es, um das Programm, nachdem es abgearbeitet worden ist, in einen definierten Endzustand zu versetzen, oder einfach Operationen solange auszuführen, wie der Mikrocontroller läuft.

Das kann man sowohl mit einer while als auch mit einer for Schleife ganz einfach realisieren:

```
while(true) { }
```

bzw.

```
for(;;) { }
```

Der Anweisungsblock wird dann „unendlich oft“ ausgeführt (bzw. bis der Mikrocontroller das Reset Signal erhält, oder man mit dem Befehl „break“ die Schleife beendet).

Der Vollständigkeit halber kann man noch die do-while Schleife erwähnen, dabei handelt es sich um eine Variante der normalen while-Schleife. Der Unterschied ist, dass der Anweisungsblock mindestens einmal ausgeführt wird, auch wenn die Bedingung nicht erfüllt ist.

Der Aufbau ist folgender:

```
do
{
	<Anweisungsblock>
}
while(<Bedingung>);
```

Hier das Semikolon am Ende nicht vergessen! (Bei normalen Schleifen gehört da natürlich keins hin!)

-

Funktionen

Ein sehr wichtiges Sprachelement sind Funktionen. Wir haben ja weiter oben auch schon mehrere solcher Funktionen gesehen und verwendet. Beispielsweise die Funktionen „writeString" und „writeInteger" und natürlich die Main Funktion.

Funktionen sind immer nützlich, wenn bestimmte Programmsequenzen in mehreren Teilen des Programms unverändert verwendet werden können – gute Beispiele sind z.B. die ganzen Textausgabefunktionen, die wir oben schon oft verwendet haben. Es wäre natürlich sehr umständlich wenn man diese identischen Programmteile immer an diejenigen Stellen kopieren müsste, wo man sie braucht und man würde unnötigerweise Speicherplatz verschwenden. Nebenbei kann man mit Funktionen Änderungen an zentraler Stelle durchführen und muss das nicht an mehreren Stellen tun. Auch kann ein Programm deutlich übersichtlicher werden, wenn man es in mehrere Funktionen aufteilt.

Deshalb kann man in C Programmsequenzen zu Funktionen zusammenfassen. Diese haben immer folgenden Aufbau:

```
<Rückgabetyp> <Funktionsname> (<Parameter 1>, <Parameter 2>, ... <Parameter n>)
{
<Programmsequenz>
}
```

Damit man sich das besser vorstellen kann, direkt mal ein kleines Beispiel mit zwei einfachen Funktionen und der schon bekannten Main Funktion:

```
8     void someLittleFunction(void)
9     {
10          writeString("[Funktion 1]\n");
11    }
12
13    void someOtherFunction(void)
14    {
15          writeString("[Funktion 2 - mal was anderes]\n");
16    }
17
18    int main(void)
19    {
20          initRobotBase(); // Beim caterpillar das hier immer als erstes aufrufen!
21
22          // Ein paar Funktionsaufrufe:
23          someLittleFunction();
24          someOtherFunction();
25          someLittleFunction();
26          someOtherFunction();
27          someOtherFunction();
28          return 0;
29    }
```

Die Ausgabe des Programms wäre folgende:

```
[Funktion 1]
[Funktion 2 — mal was anderes]
[Funktion 1]
[Funktion 2 — mal was anderes]
[Funktion 2 — mal was anderes]
```

Die Main Funktion dient als Einsprungpunkt ins Programm. Sie wird in jedem C Programm zuerst aufgerufen und MUSS somit auch immer vorhanden sein.

In unserer obigen Main Funktion wird zuerst die initRobotBase Funktion aus der CaterpillarLibrary aufgerufen, die den Mikrocontroller initialisiert (muss man beim Caterpillar immer als erstes in der Main Funktion aufrufen, sonst funktionieren viele Sachen nicht richtig!). Diese ist vom Prinzip her genau so aufgebaut wie die anderen beiden Funktionen im Listing. Anschließend werden nacheinander ein paar mal die beiden gerade definierten Funktionen aufgerufen und der Programmcode in den Funktionen ausgeführt. Nun kann man Funktionen nicht nur wie oben im Beispiel gezeigt definieren, sondern auch noch Parameter und Rückgabewerte benutzten. Im obigen Beispielprogramm wird als Parameter und Rückgabetyp „void" angegeben, was soviel wie „leer" heisst. Das bedeutet einfach, dass diese Funktionen keinen Rückgabewert und keine Parameter haben. Man kann viele Parameter für eine Funktion definieren. Die Parameter werden dabei durch Kommata voneinander getrennt.

Ein Beispiel:

```
1     void outputSomething(uint8_t something)
2     {
3           writeString("[Der Funktion wurde folgender Wert übergeben: ");
4           writeInteger(something, DEC);
5           writeString("]\n");
6     }
7     uint8_t calculate(uint8_t param1, uint8_t param2)
8     {
9           writeString("[CALC]\n");
10          return (param1 + param2);
11    }
12
13
14    int main(void)
15    {
16          initRobotBase();
17
18          // Ein paar Funktionsaufrufe:
19          outputSomething(199);
20          outputSomething(10);
21          outputSomething(255);
22
23          uint8_t result = calculate(10, 30);
24          outputSomething(result);
25          return 0;
26    }
```

Ausgabe:

```
[Der Funktion wurde folgender Wert übergeben: 199]
[Der Funktion wurde folgender Wert übergeben: 10]
[Der Funktion wurde folgender Wert übergeben: 255]
[CALC]
[Der Funktion wurde folgender Wert übergeben: 40]
```

Die Caterpillar Library enthält ebenfalls sehr viele verschiedene Funktionen. Schauen Sie sich einfach mal einige davon und die Beispielprogramme an, dann wird das Prinzip sehr schnell klar.

Arrays, Zeichenketten, Zeiger ...

Es gäbe noch viel mehr Sprachelemente und Details zu besprechen, aber hier müssen wir auf die angegebene Literatur verweisen. (Infos zu Zeigern finden Sie z.B. hier:

http://www.galileocomputing.de/openbook/c_von_a_bis_z/c_014_000.htm

Das meiste davon braucht man auch gar nicht, um die Beispielprogramme verstehen zu können. Hier geben wir nur ein paar Beispiele und Begriffe an, um einen Überblick darüber zu geben. Eine sonderlich ausführliche Beschreibung ist das allerdings nicht.

Zunächst zu Arrays. In einem Array (="Feld") kann man eine vorgegebene Anzahl von Elementen eines bestimmten Datentyps speichern. So könnte man z.B. folgendermaßen ein Array mit 10 Bytes anlegen:

```
uint8_t unserTollesArray[10];
```

Schon hat man quasi 10 Variablen mit gleichem Datentyp deklariert, die man nun über einen Index ansprechen kann:

```
unserTollesArray[0] = 8;
unserTollesArray[2] = 234;
unserTollesArray[9] = 45;
```

Jedes dieser Elemente kann man wie eine normale Variable behandeln.

Achtung: *Der Index startet immer bei 0, und wenn man ein Array mit n Elementen angelegt hat, geht der Index von 0 bis n-1 ! Also bei 10 Elementen z.B. von 0 bis 9.*

Arrays ist sehr nützlich, wenn man viele Werte gleichen Typs speichern muss. Man kann diese auch in einer Schleife der Reihe nach durchlaufen:

```
uint8_t i;
for(i = 0; i < 10; i++)
		writeInteger(unserTollesArray[i],DEC);
```

So werden der Reihe nach alle Elemente im Array ausgegeben (hier natürlich ohne Trennzeichen oder Zeilenumbrüche dazwischen). Analog dazu kann man auch ein Array in einer Schleife mit Werten beschreiben.

Ganz ähnlich sind Zeichenketten in C aufgebaut. Normale Zeichenketten sind stets im ASCII Code kodiert, wofür man pro Zeichen ein Byte benötigt. Zeichenketten sind in C nichts anderes als Arrays, die man eben als Zeichenketten interpretieren kann. Man kann z.B. folgendes schreiben:

```
uint8_t eineZeichenkette[16] = "abcdefghijklmno";
```

Und schon hat man die in Klammern stehende Zeichenkette im Speicher abgelegt.

Wir haben weiter oben auch schon ein paar UART Funktionen verwendet, die Zeichenketten über die serielle Schnittstelle ausgeben. Die Zeichenketten sind dabei nur Arrays. Der Funktion wird allerdings kein komplettes Array übergeben, sondern nur die Adresse des ersten Elements in dem Array. Die Variable, die diese Adresse enthält nennt man „Zeiger" (engl. Pointer). Um einen Zeiger auf ein bestimmtes Array Element zu erzeugen schreibt man &unserTollesArray[x] wobei x das Element ist, auf das der Zeiger zeigen soll. Diese Schreibweise wird ab und an in den Beispielprogrammen verwendet. Beispielsweise so:

```
uint8_t * zeigerAufEinElement = &eineZeichenkette[4];
```

Das brauchen Sie aber erstmal noch nicht, um die Programmbeispiele zu verstehen und eigene Programme zu schreiben. Wird hier nur der Vollständigkeit halber erwähnt.

Programmablauf und Interrupts

Wir hatten weiter oben schon gesagt, dass ein Programm generell Anweisung nach Anweisung von oben nach unten ausgeführt wird. Dann gibt es natürlich noch die normale Flusssteuerung mit Bedingungen und Schleifen, sowie Funktionen.

Neben diesem normalen Programmablauf gibt es allerdings noch sog. „Interrupts". Die verschiedenen Hardwaremodule (Timer, TWI, UART, Externe Interrupts etc.) des Mikrocontrollers können Ereignisse auslösen, auf die der Mikrocontroller so schnell wie möglich reagieren muss. Dazu springt der Mikrocontroller – (fast) egal wo er sich gerade im normalen Programm befindet – in eine sog. Interrupt Service Routine (ISR), in der dann schnellstmöglich auf das Ereignis reagiert werden kann. Keine Sorge, Sie müssen sich hier nicht noch selbst drum kümmern, für alle benötigten ISRs wurde das bereits in der CaterpillarLibrary erledigt. Aber damit Sie wissen, worum es geht und was diese seltsamen „Funktionen" in der Library sollen, erwähnen wir es hier kurz.

Die ISRs sehen wie folgt aus:

```
ISR	( <InterruptVector> )
{
	<Anweisungsblock>
}
```

z.B. für den linken Encoder am externen Interrupt Eingang 0:

```
ISR (INT0_vect)
{
	// Hier werden bei jeder Signalflanke zwei Zähler erhöht:
	mleft_dist++; // zurückgelegte Distanz
	mleft_counter++; // Geschwindigkeitsmessung
}
```

Man kann diese ISRs nicht direkt aufrufen! Das geschieht immer automatisch und meist kann man nicht vorhersagen, wann es genau passiert! Es kann zu jeder Zeit in jedem beliebigen Programmteil passieren (ausser in einem Interrupt selbst oder wenn Interrupts deaktiviert sind). Dann wird die ISR ausgeführt und danach wieder an die Stelle zurückgesprungen, an der das Programm unterbrochen wurde. Daher muss man, wenn man Interrupts einsetzt, alle zeitkritischen Dinge auch in den jeweiligen Interrupt Service Routinen erledigen. Pausen, die man anhand von Taktzyklen errechnet hat, können sonst zu lang werden, wenn sie von einem oder mehreren Interrupt Ereignissen unterbrochen werden.

In der CaterpillarLibrary werden Interrupts verwendet, um die 36kHz Modulation für die Infrarotsensorik und Kommunikation zu erzeugen. Außerdem für die RC5 Dekodierung, Timing und Delay Funktionen, um die Encoder auszuwerten, für das TWI Modul (I²CBus) und noch ein paar andere kleinere Dinge.

C-Präprozessor

Den C Präprozessor wollen wir auch noch kurz ansprechen. Wir haben diesen bereits oben verwendet – nämlich bei `#include "RobotBaseLib.h"`!

Dieser Befehl wird noch vor dem eigentlichen Übersetzen durch den GCC vom Präprozessor ausgewertet. Mit `#include "Datei"` wird der Inhalt der angegebenen Datei an dieser Stelle eingefügt. Im Falle unseres Beispielprogramms ist das die Datei CaterpillarBaseLib.h. Diese enthält Definitionen der in der CaterpillarLibrary enthaltenen Funktionen.

Das muss so gemacht werden, damit der Compiler diese Funktionen auch findet und richtig zuordnen kann.

Aber das ist natürlich noch nicht alles, was man mit dem Präprozessor machen kann. Man kann auch Konstanten (also feste nicht veränderbare Werte) definieren:

```
#define DAS_IST_EINE_KONSTANTE 123
```

Das würde die Konstante: „DAS_IST_EINE_KONSTANTE“ mit dem Wert „123“ definieren. Der Präprozessor ersetzt einfach jedes Vorkommen dieser Konstanten durch den Wert (eigentlich ist es Textersatz). Also würde z.B. hier:

```
writeInteger(DAS_IST_EINE_KONSTANTE,DEC);
```

das „DAS_IST_EINE_KONSTANTE“ durch „123“ ersetzt und wäre somit identisch zu:

```
writeInteger(123,DEC);
```

(übrigens ist auch das „DEC“ von writeInteger nur eine so definierte Konstante – hier mit dem Wert 10 – für das dezimale Zahlensystem.)

Auch einfache If-Bedingungen kennt der Präprozessor:

```
1     #define DEBUG
2
3     void someFunction(void)
4     {
5           // Mach irgendwas
6           #ifdef DEBUG
7                 writeString_P("someFunction wurde ausgeführt!");
8           #endif
9     }
```

Der Text würde hier nur ausgegeben, wenn DEBUG definiert wurde (es muss kein Wert zugewiesen werden – einfach nur definieren). Nützlich, um z.B. diverse Ausgaben, die man nur für die Fehlersuche braucht, nur bei Bedarf zu aktivieren. Wenn DEBUG im obigen Beispiel nicht definiert wäre, würde der Präprozessor den Inhalt von Zeile 7 gar nicht an den Compiler weiterleiten.

In der CaterpillarLibrary benötigen wir oft auch Makros – diese werden ebenfalls per #define definiert, man kann ihnen aber Parameter übergeben, ähnlich wie bei Funktionen.

z.B. so:

```
#define setStopwatch1(VALUE) stopwatches.watch1 = (VALUE)
```

Das lässt sich dann wie eine normale Funktion aufrufen (`setStopwatch1(100);`).

Eine wichtige Sache ist noch, dass man normalerweise hinter Präprozessor Definitionen <u>kein</u> Semikolon schreibt!

Makefiles

Das Tool „Make“ nimmt uns eine ganze Menge Arbeit ab, denn normalerweise müsste man so einiges in die Kommandozeile eintippen, um ein C Programm zu kompilieren.

Make verarbeitet ein sog. „Makefile“, das alle Befehlssequenzen und Informationen enthält, die zum Kompilieren des jeweiligen Projekts notwendig sind. Diese Makefiles werden bei jedem der Caterpillar Beispielprojekte schon fertig mitgeliefert – man kann natürlich später auch eigene Makefiles erstellen. Wie so ein Makefile im Detail aufgebaut ist, kann hier nicht beschrieben werden – das wäre zu umfangreich. Sie brauchen sich aber für alle Caterpillar Projekte ohnehin nur um die folgenden vier Einträge Gedanken zu machen – der Rest ist für Einsteiger erstmal uninteressant:

TARGET = programmName

CATERPILLAR_LIB_PATH=../../CaterpillarBaseLib

Caterpillar_LIB_PATH_OTHERS=$(CAT_LIB_PATH)/CaterpillarBase (CAT_LIB_PATH)/CaterpillarI2C)
SRC += $(Caterpillar_LIB_PATH)/CaterpillarBase/CaterpillarBaseLib.c
SRC += $(Caterpillar_LIB_PATH)/CaterpillarBase/CaterpillarBaseLib.c
SRC += $(Caterpillar_LIB_PATH)/CaterpillarI2C/I2C_Yeti_Display.c

In unseren Makefiles sind dazwischen noch einige Kommentarzeilen mit Erläuterungen und Hinweisen zu finden. Kommentare fangen in Makefiles immer mit „#“ an und werden von Make nicht weiter beachtet.

Die Beispielprojekte für den Caterpillar enthalten bereits fertige Makefiles mit den passenden Einträgen. Sie müssen diese also nur dann ändern, wenn Sie z.B. neue C Dateien zum Projekt hinzufügen wollen, oder Dateien umbenennen möchten.

Beim Eintrag „TARGET“ trägt man den Namen der C Datei ein, die die Main-Funktion enthält. Hier gibt man nur den Namen an, OHNE das „.c“ am Ende! Bei vielen anderen Einträgen muss diese Endung aber angegeben werden, also immer auf die gegebenen Beispiele und Hinweise in den Kommentaren achten!

Im Eintrag „caterpillar_LIB_PATH“ müssen Sie das Verzeichnis mit den Dateien der CaterpillarLibrary angeben. Das ist i.d.R. „../CaterpillarBaselib“ oder „../../CaterpillarBaselib“ also ein relativer Pfad.(„../“ bedeutet „eine Verzeichnisebene höher“)

Bei Caterpillar_LIB_PATH_OTHERS muss man alle sonstigen Verzeichnisse angeben, die man verwendet hat. Die CaterpillarLibrary ist in mehrere Verzeichnisse unterteilt, von denen man alle die man verwendet auch angeben muss.

Und schließlich bei „SRC“ müssen Sie alle C-Dateien angeben (keine Header Dateien! Also nicht die Dateien, die mit „.h“ enden! Die werden automatisch in allen angegebenen Verzeichnissen gesucht!), die Sie neben der Datei mit der Main-Funktion verwenden wollen. Auch die Dateien aus der CaterpillarLibrary müssen hier angegeben werden, sofern diese verwendet werden sollen.

Was bedeutet eigentlich $(CAT_LIB_PATH) ? Ganz einfach: So kann man in Makefiles Variablen verwenden! Wir haben ja schon eine „Variable“ CAT_LIB_PATH definiert. Den Inhalt kann man nun mit $(<Variable>) überall nach dieser Deklaration einfügen. Ist praktisch und erspart sehr viel Tipparbeit...

Mehr brauchen Sie an den Makefiles für den Caterpillar normalerweise nicht zu ändern. Wenn Sie aber mehr wissen möchten, finden Sie hier ein ausführliches Handbuch: http://www.gnu.org/software/make/manual/

6.3. CATERPILLAR Funktionsbibliothek (CaterpillarLibrary)

Die Caterpillar Funktionsbibliothek, auch CaterpillarLibrary oder kurz CaterpillarLib genannt, bietet viele nützliche Funktionen, um die Hardware des Caterpillar anzusteuern. Über die meisten hardwarespezifischen Dinge muss man sich damit (eigentlich) keine Gedanken mehr machen. Sie müssen also natürlich nicht das über 300 Seiten starke Datenblatt des ATMEGA16 gelesen haben, um den Roboter programmieren zu können! Es ist aber sehr nützlich, wenn man trotzdem grundlegend nachvollzieht, was die CaterpillarLibrary Funktionen machen. Außerdem sind viele Funktionen in der CaterpillarLibrary absichtlich nicht ganz perfekt – so bleibt für Sie auch noch etwas zu tun!

Man könnte noch so einige Zusatzfunktionen hinzufügen und vieles optimieren! Die Funktionen der CaterpillarLibrary verstehen sich als Grundlage, auf der man aber sehr gut aufbauen kann.

In diesem Abschnitt beschreiben wir die wichtigsten Funktionen und geben kleine Beispiele. Wer weitere Informationen dazu benötigt, sollte die Kommentare in den Dateien der Library lesen und sich die Funktionen und Beispiele genau ansehen und nachvollziehen.

Mikrocontroller initialisieren

void initRobotBase(void)

Diese Funktion müssen Sie IMMER als erstes in der Main Funktion aufrufen!

Sie initialisiert die Hardwaremodule des Mikrocontrollers. Nur wenn Sie diese Funktion als erstes aufrufen, wird der Mikrocontroller so arbeiten, wie wir es für den Caterpillar benötigen! Ein Teil wird zwar schon vom Bootloader initialisiert, aber nicht alles.

Beispiel:
```
1       #include "CaterpillarBaseLib.h"
2
3       int main(void)
4       {
5               initRobotBase(); // Initialisierung – IMMER ALS ERSTES AUFRUFEN!
6
7       // [...] Programmcode...
8
9               while(true); // Endlosschleife
10              return 0;
11      }
12
```

Jedes Caterpillar Programm muss mindestens so ausschauen! Die Endlosschleife in Zeile 9 ist notwendig, um ein definiertes Ende des Programms zu garantieren! Ohne diese Schleife könnte sich das Programm anders verhalten als erwartet!

Nur um das zu verdeutlichen: Normalerweise führt man in der Endlosschleife einen eigenen Programmcode aus, also würde man hier in Zeile 9 das Semikolon löschen und stattdessen einen Block (zwei geschweifte Klammern) einfügen, in den das eigene Programm kommt. Vor der Main Funktion (also Zeile 3) kann man eigene Funktionen definieren, die man dann aus der Hauptschleife beliebig oft aufrufen kann.

UART Funktionen (serielle Schnittstelle)

Wir haben im C-Crashkurs oben schon einige Funktionen aus der CaterpillarLibrary verwendet, vor allem die UART Funktionen. Mit diesen Funktionen kann man Textnachrichten über die serielle Schnittstelle des Roboters an den PC (oder andere Mikrocontroller) schicken und auch von diesem empfangen.

Senden von Daten über die serielle Schnittstelle

```
void writeChar(char ch)
```

Diese Funktion schickt ein einzelnes 8-Bit ASCII Zeichen über die serielle Schnittstelle.
Die Anwendung ist sehr einfach:

```
writeChar('A');
writeChar('B');
writeChar('C');
```

Gibt „ABC“ aus. Man kann auch direkt ASCII Codes übertragen:

```
writeChar(65);
writeChar(66);
writeChar(67);
```

Das ergibt im Terminal ebenfalls die Ausgabe „ABC“ - jedes ASCII Zeichen ist schließlich einer bestimmten Nummer zugeordnet, das 'A' z.B. der 65. Mit einer angepassten Kommunikationssoftware könnte man aber auch die reinen binären Werte interpretieren.

Oft braucht man auch:

```
writeChar('\n');
```

um eine neue Zeile im Terminal anzufangen.

`void writeString(char *string)`
und `writeString_P(STRING)`

Diese Funktionen sind sehr wichtig für die Fehlersuche in Programmen, denn hiermit kann man beliebige Textnachrichten an den PC schicken. Für Datenübertragungen kann man sie natürlich ebenso verwenden.

Der Unterschied zwischen writeString und writeString_**P** besteht darin, dass bei Verwendung von writeString_P die Texte nur im Flash-ROM (**P**rogram Memory) abgelegt werden und auch von dort gelesen werden, bei writeString jedoch zusätzlich in den Arbeitsspeicher geladen werden und somit doppelt Speicherplatz verbraucht wird. Und vom Arbeitsspeicher haben wir nunmal nur 2KB zur Verfügung. Wenn es nur um die Ausgabe von festem und nicht veränderlichem Text geht, sollte man deshalb immer writeString_P verwenden. Wenn man dynamische Daten ausgeben will, die ohnehin im RAM vorliegen, **muss** man natürlich das normale writeString verwenden.

Auch hier ist die Anwendung denkbar einfach:

```
writeString("ABCDEFG");
```

Gibt „ABCDEFG“ aus, belegt aber für die Zeichenkette auch Arbeitsspeicher.

```
writeString_P("ABCDEFG");
```

Gibt ebenfalls „ABCDEFG“ aus, aber ohne dabei unnötig RAM zu belegen!

```
void writeStringLength(char *data, uint8_t length, uint8_t offset);
```

Wenn man Texte mit einstellbarer Länge (length) und Anfangsposition (offset) ausgeben möchte, kann man diese Funktion verwenden.

Ein Beispiel:

```
writeStringLength("ABCDEFG", 3, 1);
```

Ausgabe: „BCD“

```
writeStringLength("ABCDEFG", 2, 4);
```

Ausgabe: „EF“

Diese Funktion belegt allerdings auch RAM Speicher und ist eigentlich nur für dynamische Textausgabe gedacht (wird z.B. von writeIntegerLength verwendet...).

```
void writeInteger(int16_t number, uint8_t base);
```

Diese sehr nützliche Funktion gibt Zahlenwerte als ASCII Text aus! Wir haben ja oben schon gesehen, dass z.B. writeChar(65) ein 'A' ausgibt und nicht 65...

Daher braucht man so eine Konvertierungsfunktion.

Beispiel:

```
writeInteger(139, DEC);
```

Ausgabe: „139“

```
writeInteger(25532, DEC);
```

Ausgabe: „25532“

Man kann den gesamten 16bit Wertebereich mit Vorzeichen von -32768 bis 32767 ausgeben. Wer größere Zahlen braucht, muss die Funktion anpassen oder besser eine eigene Funktion schreiben!

Was soll das „DEC“ als zweiter Parameter? Ganz einfach: Das bedeutet, das die Ausgabe als Dezimalzahl (engl. DECimal) erfolgt. Es gibt aber noch andere Zahlensysteme als das Dezimalsystem mit Basis 10. So kann man die Werte auch binär (BIN, Basis 2), oktal (OCT, Basis 8) oder hexadezimal (HEX, Basis 16) ausgeben.

Beispiele:

```
writeInteger(255, DEC);
```

Ausgabe: „255“

```
writeInteger(255, HEX);
```

Ausgabe: „FF“

```
writeInteger(255, OCT);
```

Ausgabe: „377“

```
writeInteger(255, BIN);
```

Ausgabe: „11111111“

Das kann für viele Anwendungen sehr nützlich sein – vor allem HEX und BIN, denn hier sieht man sofort ohne Kopfrechnen, wie die einzelnen Bits in dem Zahlenwert angeordnet sind.

```
void writeIntegerLength(uint16_t number, uint8_t base, uint8_t length);
```

Eine Variante von writeInteger. Hier kann zusätzlich die Anzahl der Stellen (length), die ausgegeben werden soll, angegeben werden. Ist eine Zahl kürzer als die angegebene Stellenzahl, werden führende Nullen angefügt. Ist sie länger, werden nur die letzten Stellen dargestellt.

Beispiele:

```
writeIntegerLength( 2340, DEC, 5);
```

Ausgabe: „02340"

```
writeIntegerLength( 2340, DEC, 8);
```

Ausgabe: „00002340"

```
writeIntegerLength( 2340, DEC, 2);
```

Ausgabe: „40"

```
writeIntegerLength( 254, BIN, 12);
```

Ausgabe: „000011111110"

Empfangen von Daten über die serielle Schnittstelle

Der Empfang von Daten über die serielle Schnittstelle läuft nun komplett Interrupt basiert ab. Die empfangenen Daten werden automatisch im Hintergrund in einem sog. Ringpuffer gespeichert.

Einzelne empfangene Bytes/Zeichen kann man mit der Funktion

```
char readChar(void)
```

aus dem Ringpuffer lesen. Ruft man diese Funktion auf, wird das jeweils nächste verfügbare Zeichen zurückgegeben und aus dem Ringpuffer gelöscht.

Ist der Ringpuffer leer, gibt die Funktion 0 zurück. Allerdings sollte man vor jedem Aufruf mit der Funktion

```
uint8_t getBufferLength(void)
```

überprüfen, wieviele neue Zeichen noch im Ringpuffer vorhanden sind.

Mehrere Zeichen hintereinander können mit der Funktion

```
uint8_t readChars(char *buf, uint8_t numberOfChars)
```

aus dem Puffer abgerufen werden. Als Parameter übergibt man der Funktion einen Zeiger auf ein Array und die Anzahl der Zeichen, die in dieses Array kopiert werden sollen. Die Funktion gibt die tatsächliche Anzahl kopierter Zeichen zurück. Das ist nützlich, falls nicht soviele Zeichen wie angefordert im Puffer vorhanden waren.

Sollte der Puffer komplett voll sein, werden alte Daten beim Empfang von neuen Zeichen nicht überschrieben, sondern es wird eine status Variable gesetzt (uart_status), die einen „übergelaufenen" Puffer signalisiert (engl. „Buffer overflow", UART_BUFFER_OVERFLOW). Sie sollten Ihre Programme so auslegen, dass es nicht dazu kommt. Meist ist in diesem Fall die Datenrate zu hoch oder das Programm wurde für längere Zeit mit mSleep o.ä. blockiert. Sie können auch die Größe des Ringpuffers anpassen.

Standardmäßig hat der Ringpuffer eine Größe von 32 Zeichen. In der Caterpillaruart.h Datei können Sie dies mit der UART_RECEIVE_BUFFER_SIZE Definition anpassen.

Ein größeres Programm dazu finden Sie bei den Beispielprogrammen! Das Beispiel „Example_02_UART_02" wurde an die neuen Funktionen angepasst.

Delay Funktionen (Verzögerungen und Zeitsteuerung)

Oft muss man Verzögerungen (Delays) in sein Programm einbauen, oder eine bestimmte Zeitspanne warten bevor eine bestimmte Aktion ausgeführt wird.

Auch dafür hat die CaterpillarLibrary Funktionen. Für die Verzögerungen einer der Timer des MEGA16 verwendet, um möglichst unabhängig von der sonstigen Programmausführung (Interrupts) zu sein und die Verzögerungen einigermaßen genau zu halten.

Aber man muss genau überlegen, wann man diese Funktionen verwenden kann! Wenn man die automatische Servoregelung und das ACS verwendet (wird später noch erläutert), könnte das zu Problemen führen! Dann sollte man nur ganz ganz, kurze Pausen von wenigen Millisekunden (<10ms) einfügen! Das Problem kann aber leicht umgangen werden, wenn man die Stopwatches verwendet, die direkt im Anschluss beschrieben werden.

void sleep(uint8_t time)

Mit dieser Funktion hält man den normalen Programmablauf für eine gewisse Zeit an. Die Zeitspanne wird dabei in Schritten von 100µs übergeben (100µs = 0.1ms = 0.0001s also für die menschliche Wahrnehmung sehr kurz...). Da wir eine 8 bit Variable benutzten, ist die maximale Zeitspanne 25500µs = 25.5ms. Interrupt Ereignisse werden trotzdem weiter behandelt – es wird nur der normale Programmablauf unterbrochen.

Beispiele:

```
sleep(1); // 100µs Pause
sleep(10); // 1ms Pause
sleep(255); // 25.5ms Pause
```

void mSleep(uint16_t time)

Wenn man längere Pausen benötigt, kann man mSleep verwenden, denn hier wird die Zeitspanne in Millisekunden angegeben. Die maximale Zeitspanne beträgt 65535ms, oder 65.5 Sekunden.

Beispiele:

```
mSleep(1); // 1ms Pause
mSleep(100); // 100ms Pause
mSleep(1000); // 1000ms = 1s Pause
mSleep(65535); // 65.5 Sekunden Pause
```

Stopwatches

Das Problem bei diesen normalen Delay Funktionen ist - wie oben schon gesagt - dass sie den normalen Programmablauf komplett unterbrechen. Das ist aber nicht immer erwünscht, denn oft muss nur ein Teil des Programms für eine bestimmte Zeit warten, während andere Dinge weiterlaufen sollen...

Das ist der größte Vorteil daran, Hardwaretimer zu verwenden, denn diese laufen unabhängig vom restlichen Programmablauf. Die CaterpillarLibrary implementiert mit den Timern universell verwendbare „Stopwatches". Diese Bezeichnung ist nicht allgemein üblich, der Autor fand den Namen aber passend, weil das ganz ähnlich wie reale Stoppuhren funktioniert. Die Stopwatches machen viele Aufgaben einfacher. Normalerweise schreibt man solche Timing Funktionen speziell auf das jeweilige Problem zugeschnitten, die CaterpillarLibrary bietet das hier etwas universeller und einfacher an, so dass man es für viele verschiedene Dinge gleichermaßen benutzen kann.

Mit den Stopwatches kann man viele Aufgaben quasi simultan erledigen – zumindest wirkt es für den Betrachter von außen so.

Es stehen acht 16bit Stopwatches (Stopwatch1 bis Stopwatch8) zur Verfügung. Diese kann man starten, stoppen, setzen und auslesen. Die Auflösung beträgt eine Millisekunde, wie bei der mSleep Funktion. Das bedeutet, dass jede Stopwatch ihren Zählstand jede Millisekunde um 1 erhöht. Für sehr zeitkritische Dinge ist das aber nicht geeignet, da die Abfragen ob eine Stopwatch einen bestimmten Wert erreicht hat, meist nicht exakt zu diesem Zeitpunkt erfolgen.

Wie man diese Funktionen anwendet, sieht man gut am folgenden kleinen Beispielprogramm von Rp6 aber die Stopwatchfunktion beim Caterpillar sieht etwa gleich aus (S. CaterpillarExamples Example_14_stopwatch):

```
1     #include "RP6RobotBaseLib.h"
2
3     int main(void)
4     {
5           initRobotBase(); // Mikrocontroller initialisieren
6           writeString_P("\RP6 Stopwatches Demo Programm\n");
7           writeString_P("___________________________\n\n");
8
9           startStopwatch1(); // Stopwatch1 starten!
10          startStopwatch2(); // Stopwatch2 starten!
11
12          uint8_t counter = 0;
13          uint8_t runningLight = 1;
14
15          // Hauptschleife:
16          while(true)
17          {
18                // Ein kleines LED Lauflicht:
19                if(getStopwatch1() > 100) // Sind 100ms (= 0.1s) vergangen?
20                {
21                setLEDs(runningLight); // LEDs setzen
22                runningLight <<= 1; // Nächste LED (shift Operation)
23                if(runningLight > 32) // Letzte LED?
24                runningLight = 1; // Ja, also wieder von vorn beginnen!
25                setStopwatch1(0); // Stopwatch1 auf 0 zurücksetzen
26                }
27
28                // Einen Zählerstand auf dem Terminal ausgeben:
29                if(getStopwatch2() > 1000) // Sind 1000ms (= 1s) vergangen?
30                {
31                writeString_P("CNT:");
32                writeInteger(counter, DEC); // Zählerstand ausgeben
33                writeChar('\n');
34                counter++; // Zähler um 1 erhöhen
35                setStopwatch2(0); // Stopwatch2 auf 0 zurücksetzen
36                }
37    }
38    return 0;
39    }
```

Das Programm ist sehr einfach gehalten. Es wird jede Sekunde ein Zählerstand über die serielle Schnittstelle ausgegeben und inkrementiert (Zeile 29 bis 36) und nebenbei noch ein Lauflicht ausgeführt (Zeile 19 bis 26), das alle 100ms weitergeschaltet wird. Dazu werden Stopwatch1 und Stopwatch2 verwendet. In Zeile 9 und 10 werden die beiden Stopwatches gestartet und fangen danach an zu zählen. In der Endlosschleife (Zeile 16 bis 37) wird dann ständig abgefragt, ob die Stopwatches einen bestimmten Zählstand überschritten haben. In Zeile 19 beispielsweise für das Lauflicht: hier wird abgefragt, ob bereits *mindestens* 100ms vergangen sind, seit die Stopwatch von 0 an zu zählen begonnen hat. Ist das der Fall, wird die nächste LED angeschaltet und die Stopwatch auf 0 zurückgesetzt (Zeile 25) und es wird erneut 100ms lang gewartet. Genauso wird bei dem Zähler verfahren, nur ist das Intervall hier 1000ms also eine Sekunde.

Das Programm finden Sie etwas ausführlicher auch auf der CD. Es ist nur ein kleines Beispiel, man kann auch viel komplexere Dinge mit den Stopwatches realisieren und diese z.B. auch bedingt starten und stoppen etc...

Beim Beispielprogramm auf der CD sind deshalb das Lauflicht und der Zähler (dort sind es sogar 3 Stück...) auch jeweils in einer eigenen Funktion untergebracht, die aus der Endlosschleife aufgerufen werden. Das macht es bei komplexeren Programmen übersichtlicher und man kann viele Programmteile so später in anderen Programmen einfacher per Copy&Paste wiederverwenden – wie z.B. so ein Lauflicht.

Es stehen mehrere Makros zur Steuerung der Stopwatches zur Verfügung.

startStopwatchX()

Startet eine bestimmte Stopwatch. Die Stopwatch wird nicht zurückgesetzt, sondern läuft einfach vom letzten Zählstand aus weiter.

Beispiele:

```
startStopwatch1();
startStopwatch2();
```

stopStopwatchX()

Hält eine bestimmte Stopwatch an.

Beispiel:

```
stopStopwatch2();
stopStopwatch1();
```

uint8_t isStopwatchXRunning()

Gibt zurück ob die Stopwatch X läuft.

Beispiel:

```
if(!isStopwatch2Running) {
// Stopwatch läuft nicht, also mache irgendwas...
}
```

setStopwatchX(uint16_t preset)

Dieses Makro setzt Stopwatch X auf den übergebenen Wert.

Beispiel:

```
setStopwatch1(2324);
setStopwatch2(0);
setStopwatch3(2);
setStopwatch4(43456);
```

getStopwatchX()

Dieses Makro gibt den Wert von Stopwatch X zurück.

Beispiel:

```
if(getStopwatch2() > 1000) { ... }
if(getStopwatch6() > 12324) { ... }
```

Seiten LEDs und Antennen

void setLEDs(uint8_t leds)

Die 4 Seiten LEDs können Sie mit dieser Funktion steuern. Am übersichtlichsten ist es, der Funktion einen binären Wert zu übergeben (sieht immer so aus: 0bxxxxxx).

Beispiel:

```
SetLEDs(0b000000); // Dieser Befehl schaltet alle LEDs aus.
SetLEDs(0b000001); // Und dieser schaltet Seite LED1 an und alle anderen aus.
SetLEDs(0b000010); // Seite LED2
SetLEDs(0b000100); // Steite LED3
SetLEDs(0b001010); // Steite LED4 und Seite LED2
SetLEDs(0b010111); // Steite LED4, Seite LED3, Seite LED2 und Seite LED1
```

Eine alternative Möglichkeit ist folgendes:

```
statusLEDs.LED1 = true; // LED1 im LED Register aktivieren
statusLEDs.LED2 = false; // LED2 im LED Register deaktivieren
updateStatusLEDs(); // Änderungen übernehmen!
```

Hier wird die Seiten LED1 angeschaltet und Seiten LED2 ausgeschaltet. Die anderen LEDs bleiben aber genau in dem Zustand, in dem sie vorher gewesen sind! Das kann nützlich/übersichtlicher sein, wenn verschiedene LEDs aus verschiedenen Funktionen heraus geändert werden sollen o.ä..

Achtung: statusLEDs.LED1 = true; schaltet NICHT direkt LED1 an! Es wird nur ein Bit in einer Variablen gesetzt! Erst updateStatusLEDs(); schaltet LED1 dann wirklich an!

Zwei der Portpins am Prozessor werden für die Antennen verwendet.

Dazu gibt es zwei vorgefertigte Funktionen.

Diese Funktion

uint8_t getHeadLeft(void)

werten die linke Antenne aus und diese

uint8_t getHeadRight(void)

die rechte Antenne.

uint8_t getTail(void)

die hintere Antenne

Die Ports, sollten nur über die vorgefertigten Funktionen angesteuert werden! Die Ports an denen die Antennen angeschlossen sind, sind zwar durch Widerstände gesichert, aber wenn sie auf Masse geschaltet werden und dann eine Antenne geschlossen wird, fließt schon ein recht hoher Strom durch den Port. Das sollte also vermieden werden.

Beispiel:

```
if(getHeadLeft() && getHeadRight()) // Beide Antennes...
else if(getHeadLeft()) // Links...
else if(getHeadRight()) // Rechts...
mSleep(50); // Die Antenne nur 20 mal pro Sekunde (20Hz) auswerten...
```

Die beiden LEDs wechseln in jedem Fall die Farbe, wenn die Antennen gedrückt werden. Das ist absichtlich so (geht nicht anders) und keineswegs eine Fehlfunktion. Normalerweise schalten die Antennen aber nur selten, deshalb stört das nicht weiter.

In C kann man auch Zeiger auf Funktionen definieren und diese dann darüber aufrufen, ohne die Funktion in der Library selbst zu definieren. Normalerweise müsste eine Funktion schon zur Zeit der Übersetzung des Programms bekannt sein und in der Caterpillar-Library eingetragen werden, damit man diese Aufrufen kann.

So kann man eigens definierte Funktionen als sog. „Event Handler", also für die Ereignisbehandlung verwenden. Wird ein Bumper gedrückt, wird innerhalb von etwa 50ms automatisch eine Funktion aufgerufen, die man extra dafür erstellt und zuvor als Event Handler registriert hat. Die Funktion muss dabei eine bestimmte Signatur haben. In diesem Fall muss es eine Funktion sein, die keinen Rückgabewert und keinen Parameter hat (beides void).

Die Funktionssignatur muss also so `void AntenneStateChanged(void)` aussehen. Den Event Handler kann man beispielsweise zu Beginn der Main Funktion registrieren. Zum Registrieren dieses Event Handlers verwendet man folgende Funktion:

```
void ANTENNE_setStateChangedHandler(void (*AntenneHandler)(void))
```

Die Notation müssen Sie nicht unbedingt genau verstehen – kurz gesagt wird hier ein Zeiger auf eine Funktion übergeben...

Hier gleich ein einfaches Beispielprogramm von RP6 dazu. Statt Bumper können Sie auch Antenne lesen:

```
1     #include "RP6RobotBaseLib.h"
2
3     // Unsere „Event Handler" Funktion für die Bumper.
4     // Die Funktion wird automatisch aus der RP6Library aufgerufen:
5     void bumpersStateChanged(void)
6     {
7           writeString_P("\nBumper Status hat sich geaendert:\n");
8
9           if(bumper_left)
10                writeString_P(" - Linker Bumper gedrueckt!\n");
11          else
12                writeString_P(" - Linker Bumper nicht gedrueckt.\n");
13          if(bumper_right)
14                writeString_P(" - Rechter Bumper gedrueckt!\n");
15          else
16                writeString_P(" - Rechter Bumper nicht gedrueckt.\n");
17    }
18
19    int main(void)
20    {
21          initRobotBase();
22
23          // Event Handler registrieren:
24          BUMPERS_setStateChangedHandler(bumpersStateChanged);
25
26          while(true)
27          {
28                task_Bumpers(); // Bumper automatisch alle 50ms auswerten
29          }
30          return 0;
31    }
```

Das Programm gibt bei jeder Änderung des Bumper-/Antennenzustands einmal den aktuellen Status beider Bumper/ Antennen aus.

Drückt man z.B. den rechten Bumper/Antennen wäre die Ausgabe:
Bumper Status hat sich geaendert:
- Linker Bumper nicht gedrueckt.
- Rechter Bumper gedrueckt!

Drückt man beide:
Bumper-/Antennenstatus hat sich geändert:
- Linker Bumper gedrueckt!
- Rechter Bumper gedrueckt!

Da man beide Bumper/Antennen nicht wirklich gleichzeitig drücken kann, könnte evtl. noch eine zusätzliche Nachricht ausgegeben werden wo nur einer der beiden gedrückt worden ist.

Wohlgemerkt wird in dem obigen Programmcode nirgendwo die Funktion bumpersStateChanged direkt aufgerufen! Das passiert automatisch aus der Caterpillar oder Rp6 Library heraus, nämlich aus der Funktion task_Bumpers bei jeder Änderung des Bumperzustands.Da task_Bumpers unsere Funktion eigentlich erstmal gar nicht kennt, geht das nur über einen Zeiger auf die entsprechende Funktion, der seinen Wert erst bei Ausführung des Aufrufs in Zeile 24 erhält.

Der Event Handler kann natürlich noch andere Aktionen auslösen, als nur Text auszugeben – z.B. könnte man den Roboter anhalten und zurücksetzen lassen. Das sollte man allerdings nicht direkt im Event Handler selbst tun, sondern in anderen Programmteilen z.B. im Event Handler einfach irgendeine Kommando-Variable setzen die man im Hauptprogramm abfragt und dann dementsprechend die Motoren steuert! Alle Event Handler sollten immer so kurz wie möglich gehalten werden! Sie können in den Event Handlern zwar alle Funktionen aus der Rp6 oder Caterpillar Library verwenden, seien Sie aber vorsichtig mit den „rotate“ und „move“ Funktionen die wir später noch besprechen werden! Benutzen Sie hier NICHT den blockierenden Modus (wenn man dann nämlich z.B. öfters auf die Bumper/Antenne drückt, funktioniert das nicht ganz so wie man es eigentlich haben wollte ;-))!

Dieses Prinzip mit den Event Handlern wird **BEI RP6** auch noch für andere Funktionen verwendet, z.B. für das ACS – da ist es sogar sehr ähnlich zu den Bumpern, denn auch dort wird bei jeder Zustandsänderung der Objektsensoren ein Event Handler aufgerufen.

Auch für den Empfang von RC5 Codes über eine Fernbedienung werden **BEI RP6** Event Handler verwendet – jedesmal wenn ein neuer RC5 Code empfangen wurde, kann ein entsprechender Event Handler aufgerufen werden.

Man muss für all diese Dinge übrigens keine Event Handler verwenden – man kann das alles auch durch einfache if Bedingungen o.ä. abfragen und dementsprechend darauf reagieren, aber mit Event Handlern werden einige Dinge einfacher und bequemer. Ist aber eher Geschmackssache was man verwendet.

Auf der CD finden Sie übrigens noch einige ausführlichere Beispielprogramme zu diesem Thema!

Bei Caterpillar lesen wir die ADC anders aus!

Unterstehendes Beispiel ist von RP6!!

ADC auslesen (Batterie-, Servomotorstrom- und weiter Sensoren)

Am ADC (Analog to Digital Convertor) sind - wie in Abschnitt 2 schon beschrieben wurde - viele Sensoren des Caterpillar angeschlossen. Natürlich bietet die CaterpillarLibrary auch hier eine Funktion, um diese auszulesen:

```
uint16_t readADC(uint8_t channel)
```

Die Funktion gibt einen 10 Bit Wert (0...1023) zurück, also benötigt man 16 Bit Variablen für die Sensorwerte.

Es stehen folgende Kanäle zur Verfügung:

ADC_BAT --> Batteriespannungs Sensor
ADC_ADC0 --> Freier ADC Kanal für eigene Sensoren
ADC_ADC1 --> Freier ADC Kanal für eigene Sensoren

Beispiele:

```
uint16_t getubat = readADC(ADC_BAT);
uint16_t adc0 = readADC(ADC_ADC0);
uint16_t adc1 = readADC(ADC_ADC1);
if(ubat < 580) writeString_P("Warnung! Batterie fast leer!");
```

Standardmäßig wird die 5V Versorgungsspannung als Referenz verwendet. Man kann aber die Funktion auch so umschreiben, dass die interne 2.56V Referenz des ATMEGA16 verwendet wird (dazu im Datenblatt vom MEGA16 nachlesen). Für die normalen Sensoren des Caterpillar wird das allerdings normalerweise nicht benötigt.

Es ist sinnvoll mehrere ADC Werte zu messen (z.B. in einem Array speichern) und dann zunächst einmal den Mittelwert davon zu bilden oder Minimum/Maximum zu bestimmen bevor man die Messwerte des ADCs auswertet. Ab und zu fängt man sich nämlich durch Störungen Messfehler ein. Im Falle der Akkuspannung ist es z.B. für einen automatischen Stromsparmodus absolut notwendig, ein paar Werte zu mitteln denn die Spannung kann sehr stark schwanken - vor allem wenn die Servomotoren laufen und wechselnd belastet werden.

Wie bei den Antennen kann man die ADC Messungen automatisieren. Es gibt auch hier schon eine kleine Funktion, die uns das Leben etwas erleichtern kann.

In diesem Fall verkürzt sie die Zeit die zum Auslesen aller Analog/Digital Wandler Kanäle im Programm benötigt wird. Ruft man diese Funktion ständig auf, werden automatisch alle ADC Kanäle der Reihe nach „im Hintergrund" (also immer wenn gerade Zeit dazu ist) ausgelesen und die Messwerte in Variablen gespeichert.

Der ADC benötigt für jede Messung etwas Zeit und mit der readADC Funktion wäre der normale Programmablauf dann für diese Zeit unterbrochen. Da die Messung aber auch ohne unser zutun läuft (der ADC ist schließlich ein Hardwaremodul im Mikrocontroller) können wir in dieser Zeit auch andere Dinge erledigen.

Es gibt für die einzelnen Kanäle folgende 16Bit Variablen, die man immer und überall im Programm verwenden kann:

ADC_BAT: adcBat
ADC_ADC0: adc0
ADC_ADC1: adc1

Sobald Sie die task_ADC() Funktion verwenden, sollten Sie nur noch diese Variablen benutzen und <u>NICHT</u> die readADC Funktion!

Beispiel von Rp6:

```
1     #include "RP6RobotBaseLib.h"
2
3     int main(void)
4           {
5           initRobotBase();
6           startStopwatch1();
7           writeString_P("\n\nKleines ADC Messprogramm...\n\n");
8           while(true)
9           {
10                if(getStopwatch1() > 300) // Alle 300ms...
11                {
12                      writeString_P("\nADC Lichtsensor Links: ");
13                      writeInteger(adcLSL, DEC);
14                      writeString_P("\nADC Lichtsensor Rechts: ");
15                      writeInteger(adcLSL, DEC);
16                      writeString_P("\nADC Akku: ");
17                      writeInteger(adcBat, DEC);
18                      writeChar('\n');
19                      if(adcBat < 600)
20                            writeString_P("Warnung! Akku ist bald leer!\n");
21                      setStopwatch1(0); // Stopwatch1 auf 0 zurücksetzen
22                }
23                task_ADC(); // ADC Auswertung - ständig aus der Hauptschleife
24          } // aufrufen! Dann sollte man aber readADC nicht
25          return 0; // mehr verwenden!
26    }
```

Hier werden alle 300ms die Messwerte der beiden Lichtsensoren und vom Akku ausgegeben. Unterschreitet die Akkuspannung *etwa* 6V, wird eine Warnung ausgegeben.

ACS – Anti Collision System

Es gibt kein Anti Collision System auf dei Caterpillar nur die Antennen! Sie können naturlich immer ein eignes IR ACS verwenden! Es gibt hier viele optionen für ein Anti Colission system zB. die AREXX IRS-003 oder Sharp GP2D12.

Die Reichweite bzw. die Sendeleistung der beiden IR LEDs des ACS, kann man mit folgenden Funktionen einstellen:

void setACSPwrOff(void) --> ACS IR LEDs aus
void setACSPwrLow(void) --> Geringe Reichweite
void setACSPwrMed(void) --> Mittlere Reichweite
void setACSPwrHigh(void) --> Maximale Reichweite

Wenn man das ACS auswerten möchte, muss man ständig aus der Hauptschleife heraus die Funktion:

void task_ACS(void)

Aufrufen, die das ACS komplett steuert. Der Rest sieht dann ganz ähnlich wie bei den Antennen aus.

Es gibt zwei Variablen:

obstacle_left und obstacle_right

welche jeweils den Wert true haben, sobald ein Hindernis detektiert wurde. Sind beide true, befindet sich das Hindernis mittig vor dem Roboter.

Wenn man möchte, kann man auch hier einen Event Handler erstellen, muss man aber nicht – also genau wie bei den Antennen.

void ACS_setStateChangedHandler(void (*acsHandler)(void))

Mit dieser Funktion registriert man den Event Handler, der folgende Signatur haben muss:
void acsStateChanged(void)

Wie die Funktion genau heisst, ist dabei übrigens fast egal.

Das folgende Beispielprogramm zeigt die Anwendung. Zuerst wird der Event Handler registriert (Zeile 44), dann die Stromversorgung zum IR Empfänger eingeschaltet (Zeile 46 – ohne das funktioniert es nicht!) und die Sendestärke der ACS IR LEDs eingestellt (Zeile 47). Unten in der Hauptschleife wird dann die Funktion task_ACS() ständig aufgerufen.

Der Rest erfolgt automatisch – die acsStateChanged Funktion wird immer aufgerufen, wenn sich der Zustand des ACS verändert hat – also wenn z.B. ein Objekt von den Sensoren entdeckt wird und wieder verschwindet. Das Programm stellt den aktuellen Zustand des ACS als Text im Terminal und mit den LEDs dar.

Wieder ein RP6 Beispiel:

```
1     #include "RP6RobotBaseLib.h"
2
3     void acsStateChanged(void)
4     {
5           writeString_P("ACS status hat sich geändert! L: ");
6
7           if(obstacle_left) // Hindernis links
8                 writeChar('o');
9           else
10                writeChar(' ');
11
12          writeString_P(" | R: ");
13
14          if(obstacle_right) // Hindernis rechts
15                writeChar('o');
16          else
17                writeChar(' ');
18
19          if(obstacle_left && obstacle_right) // Mitte?
20                writeString_P(" MITTE!");
21          writeChar('\n');
22
23          statusLEDs.LED6 = obstacle_left && obstacle_right; // Mitte?
24          statusLEDs.LED3 = statusLEDs.LED6;
25          statusLEDs.LED5 = obstacle_left; // Hindernis links
26          statusLEDs.LED4 = (!obstacle_left); // LED5 invertiert!
27          statusLEDs.LED2 = obstacle_right; // Hindernis rechts
28          statusLEDs.LED1 = (!obstacle_right); // LED2 invertiert!
29          updateStatusLEDs();
30    }
31
32    int main(void)
33    {
34          initRobotBase();
35
36          writeString_P("\nRP6 ACS - Testprogramm\n");
37          writeString_P("_____________________\n\n");
38
39          setLEDs(0b111111);
40          mSleep(1000);
41          setLEDs(0b001001);
42
43          // ACS Event Handler registrieren:
44          ACS_setStateChangedHandler(acsStateChanged);
45
46          powerON(); // ACS Empfänger einschalten (und Encoder etc.)
47          setACSPwrMed(); // ACS auf mittlere Sendeleistung stellen.
48
49          while(true)
50          {
51                task_ACS(); // ständig die task_ACS Funktion aufrufen!
52          }
53          return 0;
54    }
```

In diesem Beispiel sieht man auch nochmal sehr schön wie man die LEDs einzeln setzen kann.

Wenn man das Programm ausführt, sollte man den RP6 natürlich am Rechner angeschlossen lassen und sich die Ausgaben auf dem Bildschirm ansehen. Bewegen Sie einfach mal die Hand oder einen Gegenstand direkt vor dem Roboter hin und her!

Das ACS kann durch Störquellen in seiner Funktion beeinträchtigt werden! Bestimmte Leuchtstoffröhren und ähnliches können den Roboter quasi „blind" machen oder zumindest die Empfindlichkeit verringern. Wenn Sie also Probleme damit haben, schalten Sie erstmal alle evtl. störenden Lichtquellen aus. (Tipp: Wenn Sie den Roboter direkt vor einem Flachbildschirm oder Flachbildfernseher platziert haben, kann es auch daran liegen. Meist wird dort schließlich auch eine Leuchtstoffröhre für die Hintergrundbeleuchtung verwendet...) Die Reichweite hängt außerdem stark von den Gegenständen selbst ab, denn eine schwarze Oberfläche reflektiert das IR Licht natürlich weniger gut als eine helle weisse Oberfläche. Sehr dunkle Objekte werden vom ACS nicht immer erkannt!

So gesehen, sollte man das ACS noch mit Ultraschallsensoren oder besseren IR Sensoren unterstützen.

Unterstehendes Beispiel ist von RP6!!

IRCOMM und RC5 Funktionen

Mit dem IR Empfänger kann der Caterpillar auch IR Signale von normalen TV/Hifi Fernbedienungen empfangen. Das klappt allerdings nur, wenn die Fernbedienung im RC5 Code sendet! Die meisten Universalfernbedienungen (s.Abb.) können auf dieses Format eingestellt werden – wie das geht steht im Handbuch der jeweiligen Universalfernbedienung. Wenn in der Codetabelle nicht explizit der RC5 Code
aufgeführt wird, kann man einfach einige verschiedene Gerätehersteller durchprobieren.

Wenn die Fernbedienung im RC5 Format sendet, wird dieses Signal von der ACS Hinderniserkennung ignoriert und das ACS nur sehr wenig bis gar nicht gestört. Es kann trotzdem noch Hindernisse detektieren, möglicherweise aber etwas später da so nur die Pausen zwischen den RC5 Übertragungen genutzt werden können. Sollte die Fernbedienug allerdings nicht im RC5 Format senden, wird natürlich der gesendete Code nicht erkannt und das ACS könnte dadurch gestört werden.

Mit einer passenden Software im Mikrocontroller, kann man den Caterpillar dann mit einer Fernbedienung steuern und ihm Befehle senden!

Über das IRCOMM System kann man auch IR Signale senden. Die beiden Sendedioden vorne am Roboter sind nach oben zur Decke gerichtet. Über die Reflektion an der Zimmerdecke und anderen Gegenständen oder bei direkter Sichtverbindung nach vorn, kann man so mit anderen Robotern oder z.B. einer Basisstation kommunizieren. Das geht zwar nur relativ langsam (ein Datenpaket braucht ca. 20ms und danach eine kleine Pause), aber einfache Kommandos und einzelne Messwerte lassen sich schon damit übertragen. Die Reichweite ist natürlich etwas eingeschränkt und es funktioniert nur im selben Raum wenn die Roboter maximal etwa 2 bis 4 Meter voneinander entfernt sind (je nach Lichtverhältnissen, Hindernissen, Beschaffenheit der Zimmerdecke und den Aufbauten/Erweiterungen auf den Robotern). Die Reichweite lässt sich aber steigern, wenn man z.B. noch ein paar weitere IR LEDs hinzuschaltet (z.B. mit einem weiteren MOSFET, einem großen Kondensator und kleinem Vorwiderstand).

Aufgrund der notwendigen Synchronisation mit dem ACS ist auch für diese Aufgaben die task_ACS() Funktion zuständig. Diese muss also auch deshalb ständig in der Hauptschleife aufgerufen werden, damit auf empfangene IR Daten reagiert werden kann - und außerdem um die Übertragungen mit dem IRCOMM auszuführen!

RC5 Datenpakete enthalten immer eine Geräteadresse, den „Keycode“ und ein „Togglebit“. Die 5 Bit Geräteadresse gibt an, welches Gerät angesteuert werden soll. Bei normaler Verwendung kann man so die Fernbedienungen für TV, Videorecorder, HiFi Anlage etc. unterscheiden. Bei unserer Anwendung kann man anstattdessen z.B. verschiedene Roboter adressieren. Der 6 Bit Keycode ist bei normalen Fernbedienungen der Tastencode, aber wir können damit natürlich auch Daten übertragen. Zwar nur 6 Bit, aber 8 Bit Daten könnte man auf zwei Übertragungen aufteilen, oder noch 2 der 5 Bit Geräteadresse oder das Togglebit zweckentfremden.

Das Togglebit wird normalerweise dazu verwendet, um zu unterscheiden ob eine Taste auf der Fernbedienug dauerhaft gedrückt wird oder mehrmals hintereinander. Das Togglebit kann bei Datenübertragungen von Roboter zu Roboter aber beliebig verwendet werden.

RC5 Daten kann man mit dieser Funktion senden:

```
void IRCOMM_sendRC5(uint8_t adr, uint8_t data)
```

Wobei adr die Geräteadresse und data Keycode bzw. Daten sind. In adr kann man zusätzlich das Toggle Bit setzen, indem man das höchstwertige Bit dieses Bytes setzt. Dazu kann man die Konstante TOGGLEBIT wie folgt verwenden:

```
IRCOMM_sendRC5(12 | TOGGLEBIT, 40);
```

Dieser Befehl sendet ein RC5 Datenpaket an Adresse 12, bei gesetztem Togglebit und mit 40 als Daten.

```
IRCOMM_sendRC5(12, 40);
```

So sieht es dann aus, wenn man das Togglebit NICHT setzen will.

Der Empfang von RC5 Daten funktioniert über einen Event Handler, ähnlich wie bei den Bumpern und beim ACS. Der Event Handler wird automatisch aus der task_ACS() Funktion aufgerufen sobald neue RC5 Daten vorliegen. Beispielweise könnte man dann, wenn Tastencode 4 empfangen wurde, nach links drehen und bei Code 6 nach rechts o.ä....

In einem der Beispielprogramme wird genau das gemacht: man kann die Bewegung des Roboters komplett über eine IR Fernbedienung steuern.

Der Event Handler muss folgende Signatur haben:

```
void receiveRC5Data(RC5data_t rc5data)
```

Wie die Funktion heisst, ist natürlich auch hier fast egal!

```
void IRCOMM_setRC5DataReadyHandler(void (*rc5Handler)(RC5data_t))
```

Mit dieser Funktion kann man den zuvor definierten Event Handler registrieren.
z.B. so:

```
IRCOMM_setRC5DataReadyHandler(receiveRC5Data);
```

Danach wird immer diese Funktion aufgerufen sobald ein gültiger RC5 Code empfangen worden ist.

RC5data_t ist ein eigens definierter Datentyp, der RC5 Adressbits (Device Address), Togglebit und Keycode (bzw. die Daten) enthält. Diese sind dann ähnlich wie normale Variablen verwendbar und haben folgende Bezeichnungen:

```
rc5data.device, rc5data.toggle_bit, rc5data.key_code
```

7. Zum Abschluss

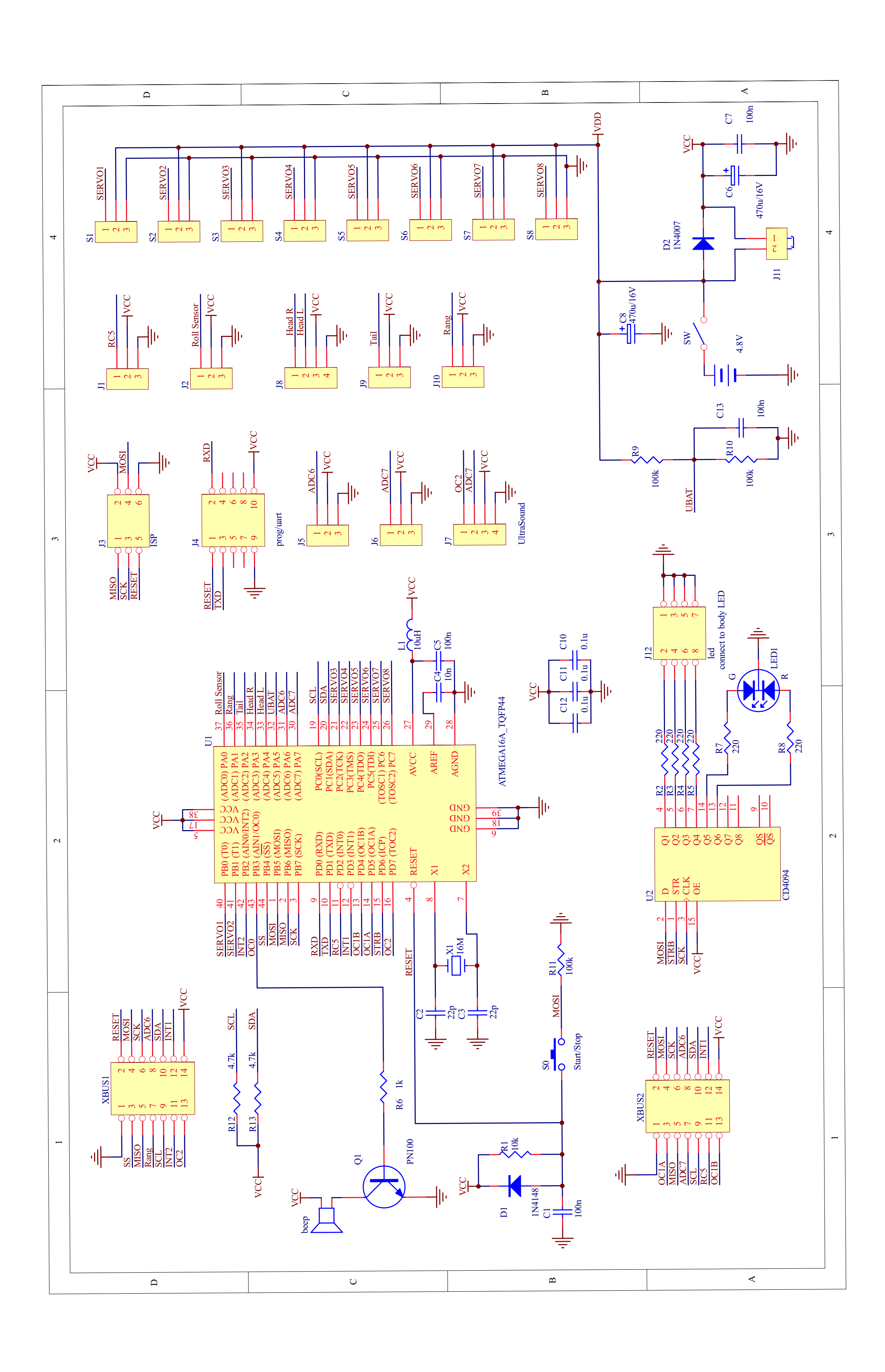
Wir hoffen, dass unsere Roboter CATTERPILLAR, ASURO und YETI Ihnen auf den Weg in die Roboterwelt geholfen haben! Wie unsere japanischen Freunde glauben auch wir, dass Roboter nach den Computern und Mobiltelefonen die nächste technologische Revolution bilden werden. Diese Revolution wird auch neue wirtschaftliche Impulse auslösen. Leider haben Japan, andere asiatische Länder und auch die USA, Europa dabei längst überholt. Im Gegensatz zu Europa beginnt der Technikunterricht in Asien bereits in der Grundschule und ist ein wichtiger Bestandteil der Ausbildung.

Als Zielsetzung bei der Entwicklung der Roboter CATERPILLAR, ASURO und YETI haben wir deshalb gewählt:

TO TRAIN A SCIENTIFIC MIND

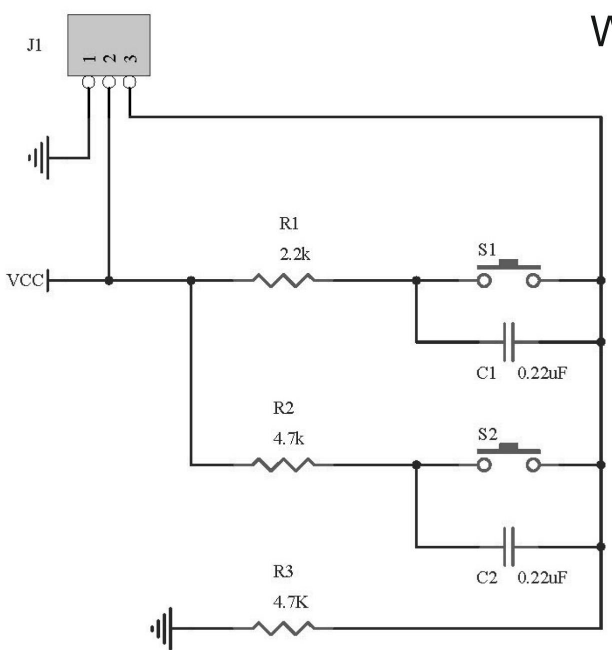
APPENDIX

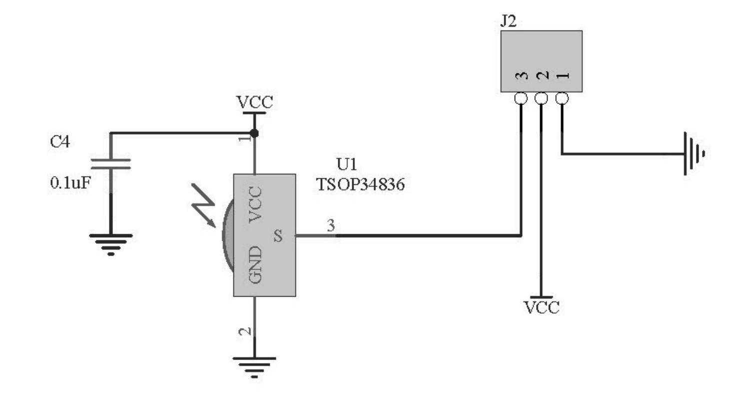
A. Caterpillar Hauptplatine



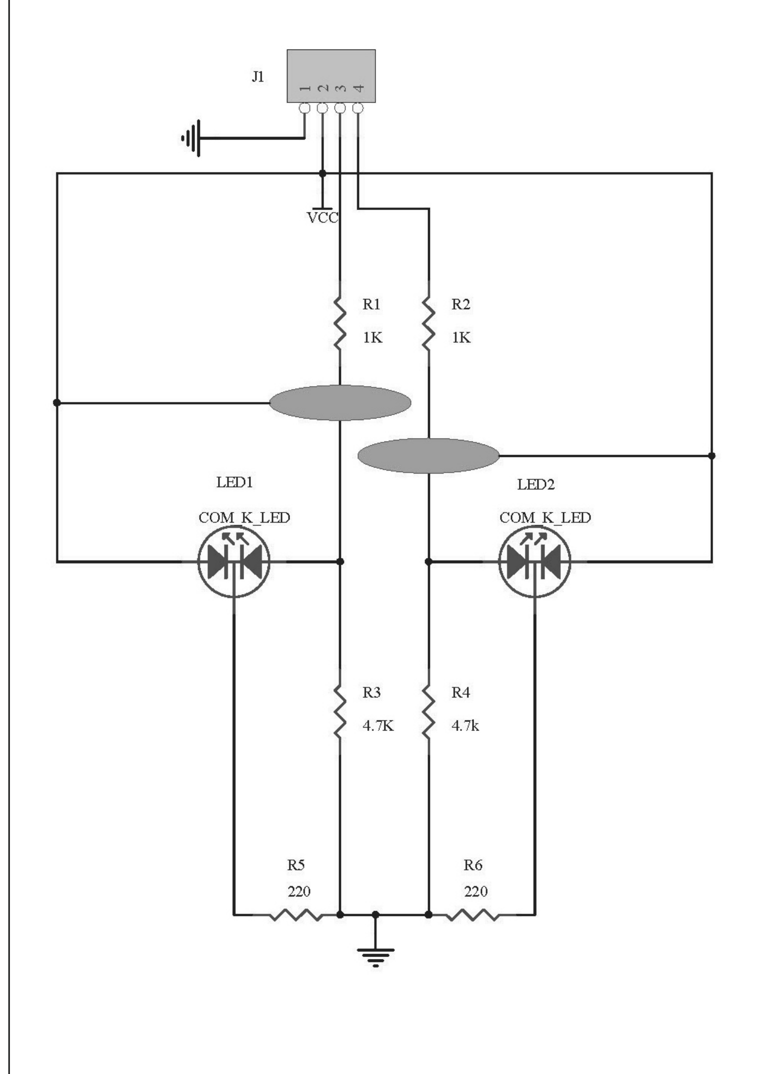
B. Caterpillar Sensoren

Winkel Sensor

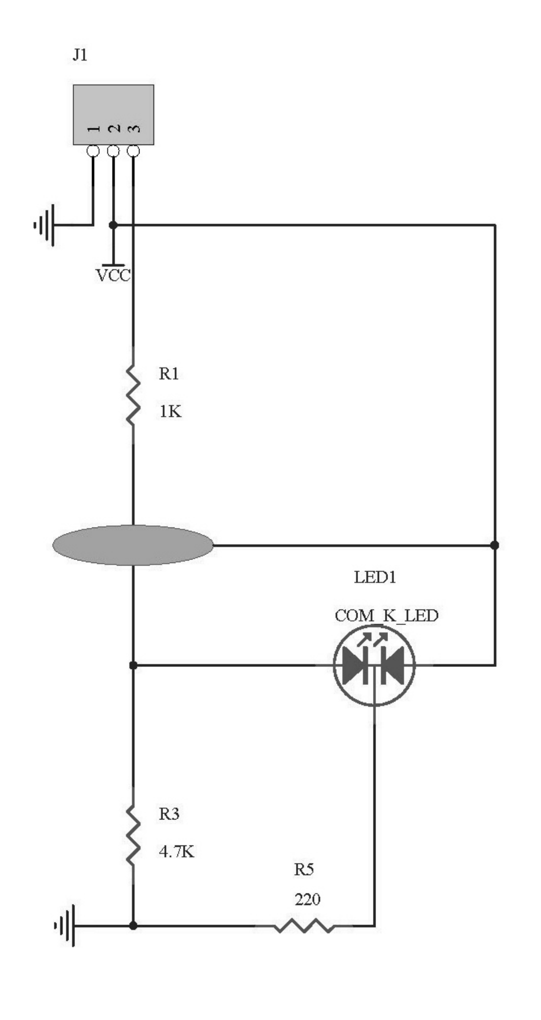


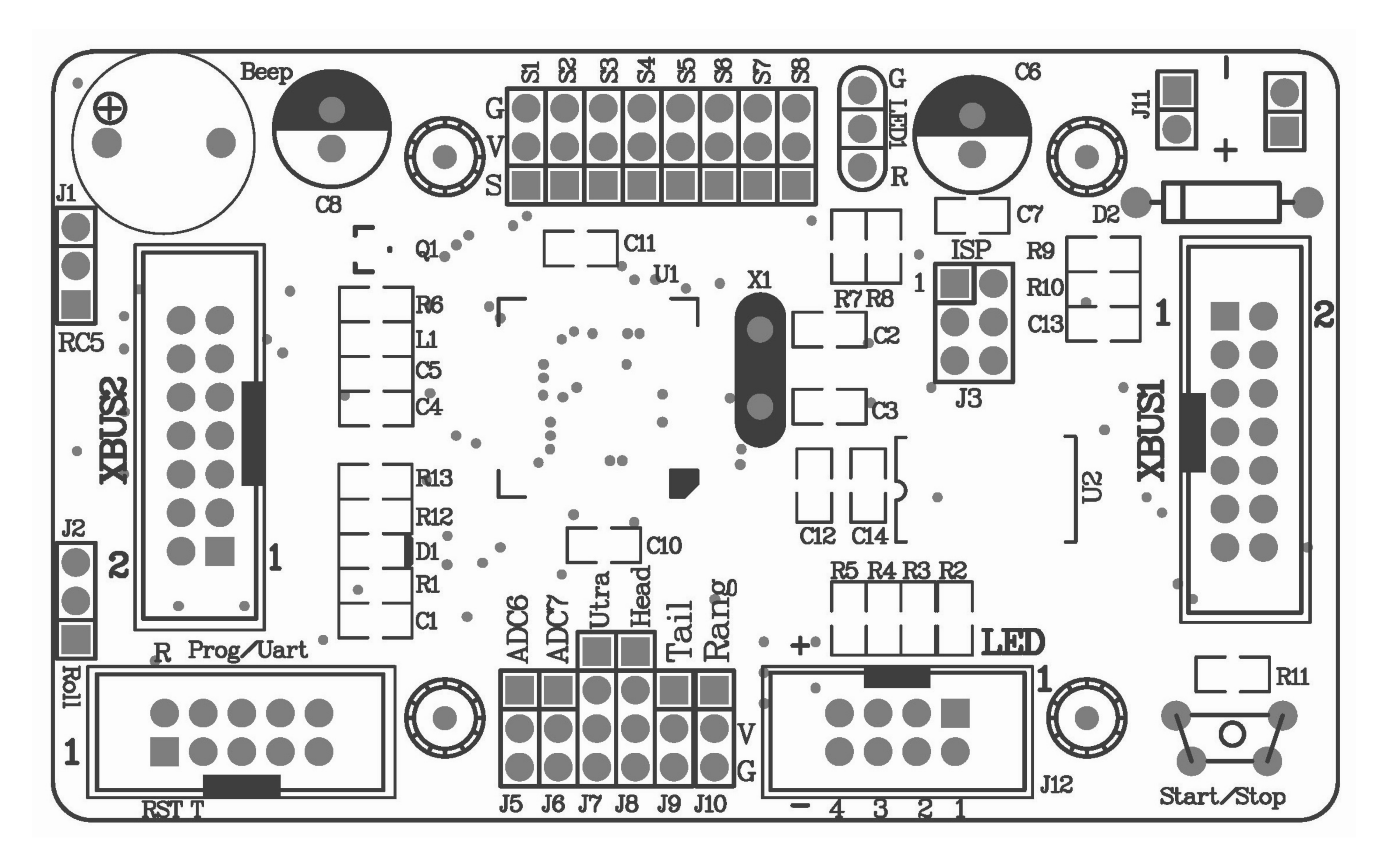


Kopf Antenne



Schwanz Antenne





Beep
S1
S2
S3
S4
S5
S6
S7
S8
G
V
S
LED1
G
R
C6
J11
C8
J1
RC5
Q1
C11
U1
X1
R7 R8
ISP
C7
D2
R9
R10
C13
J3
R6
L1
C5
C4
C2
C3
XBUS2
XBUS1
R13
R12
D1
R1
C1
C12
C14
U2
C10
J2
Roll
R Prog/Uart
RST T
ADC6
ADC7
Utra
Head
Tail
Rang
V
G
J5
J6
J7
J8
J9
J10
R5 R4 R3 R2
LED
J12
R11
Start/Stop